

Algoritmos y Estructuras de Datos II

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

Trabajo Práctico N°2

Grupo 19

Integrante	LU	Correo electrónico
Basso, Juan Cruz	627/14	jcbasso95@gmail.com
Bohe, Brian	706/14	brianbohe@gmail.com
Figari, Francisco	719/14	francisco.figari@hotmail.com
Mariotti, Ignacio	651/14	nacho692@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Nociones previas	3
2. Modulos	3
2.1. Modulo Campus Seguro	3
2.2. Campus	20
2.3. Diccionario de Nombres	25
2.3.1. Iterador de Diccionario de Nombres	29
2.3.2. Iterador de Claves de Diccionario de Nombres	32
2.4. Diccionario Logaritmico	34
2.4.1. Iterador de Diccionario Logaritmico	37
2.4.2. Iterador de Claves de Diccionario Logaritmico	38
2.5. Diccionario _H	41
2.6. Posicion	43
2.7. Matriz(α)	45
3. Tads	48
3.1. Posicion	48
3.2. Matriz	49
3.3. Diccionario Acotado	50

1. Nociones previas

- Para las complejidades se utilizan las definiciones del contexto de uso y se agregan algunas otras
 - $|n_m|$ es el nombre mas largo de estudiantes y hippies
 - $|h_m|$ es el nombre mas largo de hippies
 - $|e_m|$ es el nombre mas largo de estudiantes
 - N_a es la cantidad de agentes
 - N_h es la cantidad de hippies
 - N_e es la cantidad de estudiantes
- Algunas consideraciones de tipos son
 - *direccion* es un enumerado izq,der,arriba,abajo
 - *placa* es nat
 - *agente* es nat
 - *nombre* es string
- Al no haber un modulo de arreglos explicito en modulos basicos, pueden haber diferencias triviales en los diferentes algoritmos
- Se asume que el módulo nat tiene las siguientes funciones
 - $\text{MAX}(\text{in } a : \text{nat}, \text{in } b : \text{nat}) \rightarrow res : \text{nat}$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{if } a > b \text{ then } a \text{ else } b \text{ fi}\}$
Complejidad: $\mathcal{O}(1)$
 - $\text{MIN}(\text{in } a : \text{nat}, \text{in } b : \text{nat}) \rightarrow res : \text{nat}$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{if } a < b \text{ then } a \text{ else } b \text{ fi}\}$
Complejidad: $\mathcal{O}(1)$
 - $\text{MOD}(\text{in } a : \text{nat}, \text{in } m : \text{nat}) \rightarrow res : \text{nat}$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{modulo}(a, m)\}$
Complejidad: $\mathcal{O}(1)$
- Se asume que el módulo bool tiene la siguiente función
 - $\beta(\text{in } b : \text{bool}) \rightarrow res : \text{nat}$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{if } b \text{ then } 1 \text{ else } 0 \text{ fi}\}$
Complejidad: $\mathcal{O}(1)$

modulo : $\text{nat} \times \text{nat} \rightarrow \text{nat}$
 modulo(a, m) $\equiv \text{if } a < m \text{ then } a \text{ else modulo}(a - m) \text{ fi}$

2. Modulos

2.1. Modulo Campus Seguro

Se sigue el siguiente orden para las funciones que proveen los movimientos de estudiantes, agentes o hippies.

1. Apariciones y Movimientos
2. Enhippizacion (Estudiantes en hippies)
3. Educacion (Hippies en estudiantes)

4. Premios de capturas de hippies
5. Captura de hippies
6. Sanciones relacionadas con el evento

Interfaz

se explica con: CAMPUS SEGURO.

géneros: campusSeg.

Operaciones básicas de campusSeg

CAMPUS(**in** c : campusSeg) $\rightarrow res$: campus

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} campus(c)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve el campus de Campus Seguro

Aliasing: El campus se devuelve por referencia

ESTUDIANTES(**in** c : campusSeg) $\rightarrow res$: itClavesDiccN(nombre)

Pre $\equiv \{true\}$

Post $\equiv \{esPermutacion(Siguientes(res), secuenciarConj(estudiantes(d)))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve Iterador no modificable de todos los estudiantes actualmente en el campus

Aliasing: Si hay un cambio en los estudiantes, el iterador puede quedar invalidado

HIPPIES(**in** c : campusSeg) $\rightarrow res$: itClavesDiccN(nombre)

Pre $\equiv \{true\}$

Post $\equiv \{esPermutacion(Siguientes(res), secuenciarConj(hippies(d)))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve Iterador no modificable de todos los hippies actualmente en el campus

Aliasing: Si hay un cambio en los hippies, el iterador puede quedar invalidado

AGENTES(**in** c : campusSeg) $\rightarrow res$: itClavesDiccLog(nat)

Pre $\equiv \{true\}$

Post $\equiv \{esPermutacion(Siguientes(res), secuenciarConj(agentes(c)))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve Iterador no modificable de todos los agentes actualmente en el campus

Aliasing: Si hay un cambio en el diccionario, se va a ver reflejado en el iterador

POSESTUDIANTEYHIPPIE(**in** c : campusSeg, **in** n : nombre) $\rightarrow res$: posicion

Pre $\equiv \{n \in estudiantes(c) \cup hippies(c)\}$

Post $\equiv \{res =_{obs} posEstudianteYHippie(n, c)\}$

Complejidad: $\mathcal{O}(|n_m|)$ con siendo $|n_m|$ la longitud del nombre más largo de hippies y estudiantes

Descripción: Devuelve la posicion del estudiante o hippie

POSAGENTE(**in** c : campusSeg, **in** p : placa) $\rightarrow res$: posicion

Pre $\equiv \{pl \in agentes(c)\}$

Post $\equiv \{res =_{obs} posAgente(pl, c)\}$

Complejidad: $\mathcal{O}(1)$ en caso promedio

Descripción: Devuelve la posicion del agente

CANTSANCIONES(**in** c : campusSeg, **in** pl : placa) $\rightarrow res$: nat

Pre $\equiv \{pl \in agentes(c)\}$

Post $\equiv \{res =_{obs} cantSanciones(pl, c)\}$

Complejidad: $\mathcal{O}(1)$ en caso promedio

Descripción: Devuelve la cantidad de sanciones del agente

CANTHIPPIESATRAPADOS(**in** c : campusSeg, **in** pl : placa) $\rightarrow res$: nat

Pre $\equiv \{pl \in \text{agentes}(c)\}$

Post $\equiv \{res =_{\text{obs}} \text{cantHippiesAtrapados}(pl, c)\}$

Complejidad: $\mathcal{O}(1)$ en caso promedio

Descripción: Devuelve la cantidad de hippies que atrapo el agente

COMENZARRASTRILLAJE(**in** c : campusSeg, **in** d : dicc(agente, posicion)) $\rightarrow res$: campusSeg

Pre $\equiv \{(\forall a: \text{agente}) \text{def?}(a, d) \Rightarrow_L \text{posValida?}(\text{obtener}(a, d)) \wedge \neg \text{ocupada?}(\text{obtener}(a, d), c))\}$

$\wedge (\forall a_0, a_1: \text{agente}) ((\text{def?}(a_0, d) \wedge \text{def?}(a_1, d) \wedge a_0 \neq a_1) \Rightarrow_L \text{obtener}(a_0, d) \neq \text{obtener}(a_1, d))\}$

Post $\equiv \{res =_{\text{obs}} \text{comenzarRastrillaje}(c, d)\}$

Complejidad: $\mathcal{O}(h * a + n^2) + \mathcal{O}(n)$ prom

Descripción: Inicializa toda la estructura en funcion de los datos de entrada

INGRESARESTUDIANTE(**in/out** c : campusSeg, **in** n : nombre, **in** p : posicion)

Pre $\equiv \{c = c_0 \wedge n \notin (\text{estudiantes}(c) \cup \text{hippies}(c)) \wedge \text{esIngreso}(p, \text{campus}(c)) \wedge \neg \text{estaOcupada?}(p, c)\}$

Post $\equiv \{c = \text{ingresarEstudiante}(n, p, c_0)\}$

Complejidad: $\mathcal{O}(|n_m|)$

INGRESARHIPPIE(**in/out** c : campusSeg, **in** n : nombre, **in** p : posicion)

Pre $\equiv \{c = c_0 \wedge n \notin (\text{estudiantes}(c) \cup \text{hippies}(c)) \wedge \text{esIngreso}(p, \text{campus}(c)) \wedge \neg \text{estaOcupada?}(p, c)\}$

Post $\equiv \{c = \text{ingresarHippie}(n, p, c_0)\}$

Complejidad: $\mathcal{O}(|n_m|)$

MOVERESTUDIANTE(**in/out** c : campusSeg, **in** n : nombre, **in** d : direccion)

Pre $\equiv \{c = c_0 \wedge n \in \text{estudiantes}(c) \wedge (\text{seRetira}(n, d, c) \vee$

$(\text{posValida}(\text{proxPosicion}(\text{posEstudianteYHippie}(n, c), d, \text{campus}(c)), \text{campus}(c)) \wedge$

$\neg \text{estaOcupada?}(\text{proxPosicion}(\text{posEstudianteYHippie}(n, c), d, \text{campus}(c)), c))\}$

Post $\equiv \{c = \text{moverEstudiante}(n, d, c_0)\}$

Complejidad: $\mathcal{O}(|n_m|)$

MOVERHIPPIE(**in/out** c : campusSeg, **in** n : nombre)

Pre $\equiv \{c = c_0 \wedge n \in \text{hippies}(c) \wedge \neg \text{todasOcupadas?}(\text{vecinos}(\text{posEstudianteYHippie}(n, c), \text{campus}(c)), c)\}$

Post $\equiv \{c = \text{moverHippie}(n, c_0)\}$

Complejidad: $\mathcal{O}(|n_m|) + \mathcal{O}(N_e)$

MOVERAGENTE(**in/out** c : campusSeg, **in** a : agente)

Pre $\equiv \{c = c_0 \wedge$

$a \in \text{agentes}(c) \wedge_L \text{cant Sanciones}(a, c) \leq 3 \wedge \neg \text{todasOcupadas?}(\text{vecinos}(\text{posAgente}(a, c), \text{campus}(c)), c)\}$

Post $\equiv \{c = \text{moverAgente}(a, c_0)\}$

Complejidad: $\mathcal{O}(|n_m|) + \mathcal{O}(\log N_a) + \mathcal{O}(N_h)$

MASVIGILANTE(**in** c : campusSeg) $\rightarrow res$: placa

Pre $\equiv \{\# \text{Agentes}(c) \geq 1\}$

Post $\equiv \{res =_{\text{obs}} \text{masVigilante}(c)\}$

Complejidad: $\mathcal{O}(1)$

CONMISMASANCIONES(**in** c : campusSeg, **in** a : agente) $\rightarrow res$: conj(placa)

Pre $\equiv \{a \in \text{Agentes}(c)\}$

Post $\equiv \{res =_{\text{obs}} \text{conMismasSanciones}(a, c)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: El conjunto se devuelve por referencia y no se puede modificar

CONKSANCIONES(**in/out** c : campusSeg, **in** k : nat) $\rightarrow res$: conj(placa)

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

Post $\equiv \{res =_{\text{obs}} \text{conKsanciones}(k, c)\}$

Complejidad: $\mathcal{O}(N_a)$ o $\mathcal{O}(\log N_a)$ dependiendo si hubo cambios en las sanciones

Descripción: El conjunto se devuelve por referencia y no se puede modificar

CANTHIPPIES(**in** c : campusSeg) $\rightarrow res$: nat

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

Post $\equiv \{res =_{\text{obs}} \text{cantHippies}(c)\}$

Complejidad: $\mathcal{O}(1)$

12. mE, el booleano es true si y solo si hay un estudiante en esa posicion.
Si es true, la siguiente clave del iterador es el estudiante en esa posicion y la secuencia subyacente es permutacion de los estudiantes en estudiantes.
13. El mas vigilante es el de mayor sanciones de los agentes y, entre ellos, el de menor placa.

Para facilitar la lectura se renombra las matrices e.pos.mXX a mXX

Rep : estr \longrightarrow bool

Rep(e) \equiv true \iff

1. $\text{claves}(e.\text{hippies}) \cap \text{claves}(e.\text{estudiantes}) = \emptyset$
2. $\wedge (\forall p: \text{posicion})(\#(p, \text{armarMultiPos}(e)) \geq 1 \Rightarrow \#(p, \text{armarMultiPos}(e)) = 1 \wedge \text{posValida}(p, e.\text{cp}) \wedge_L \neg \text{ocupada?}(p, e.\text{cp}))$
3. $\wedge (\text{claves}(e.\text{agentes}) = \text{claves}(e.\text{agentesRapido}) \wedge_L (\forall p: \text{placa}) (\text{def?}(p, e.\text{agentes}) \Rightarrow_L \pi_1(\text{Siguiente}(\text{obtener}(p, e.\text{agentesRapido}))) = \text{obtener}(p, e.\text{agentes})) \wedge \text{esPermutacion}(\text{SecuSuby}(\text{obtener}(p, e.\text{agentesRapido})), \text{secuenciarDic}(e.\text{agentes})))$
4. $\wedge (\forall c: \text{nat})(\forall s: \text{conj}(\text{placa})) (\text{esta?}(\langle c, s \rangle, e.\text{sanciones.lista}) \Rightarrow (s \subseteq \text{claves}(e.\text{agentes}) \wedge \neg \emptyset?(s) \wedge_L (\forall pl: \text{placa}) pl \in s \iff \text{obtener}(pl, e.\text{agentes}).\text{sanciones} = c \wedge \text{obtener}(pl, e.\text{agentes}).\text{pl} = pl))$
5. $(\forall i: \text{nat})(i \leq \text{longitud}(e.\text{lista}) - 2) e.\text{sanciones.lista}[i].c < e.\text{sanciones.lista}[i + 1].c$
6. $\neg e.\text{sanciones.actualizar} \Rightarrow (\forall c: \text{nat})(\forall s: \text{conj}(\text{nat})) (\text{esta?}(\langle c, s \rangle, e.\text{sanciones.lista}) \iff \text{def?}(c, e.\text{sanciones.rapidas}) \wedge_L s = \text{obtener}(c, e.\text{sanciones.rapidas}))$
7. $\wedge (\forall p: \text{placa}) \text{def?}(p, e.\text{agentes}) \Rightarrow_L \text{SecuSuby}(\text{obtener}(p, e.\text{agentes}).\text{kSanc}) = \text{ordenar}\pi_1(e.\text{sanciones.lista}) \wedge \pi_1(\text{Siguiente}(\text{obtener}(p, e.\text{agentes}).\text{kSanc})) = \text{obtener}(p, e.\text{agentes}).\text{pl}$
8. $\wedge (\forall c: \text{nat})(\forall s: \text{conj}(\text{placa}))(\forall pl: \text{placa}) (\text{esta?}(\langle c, s \rangle, e.\text{sanciones.lista}) \wedge (pl \in s \wedge \text{def?}(pl, e.\text{agentes}))) \Rightarrow_L \text{esPermutacion}(\text{secuenciarConj}(s), \text{SecuSuby}(\text{obtener}(pl, e.\text{agentes}).\text{mismasSanc})) \wedge \text{siguiente}(\text{obtener}(pl, e.\text{agentes}).\text{mismasSanc}) = pl$
9. $\wedge \text{Ancho}(\text{mAg}) = \text{columnas}(e.\text{campus}) + 1 \wedge \text{Alto}(\text{mAg}) = \text{filas}(e.\text{campus}) + 1 \wedge \text{Ancho}(\text{mH}) = \text{Ancho}(\text{mAg}) \wedge \text{Alto}(\text{mH}) = \text{Alto}(\text{mAg}) \wedge \text{Ancho}(\text{mE}) = \text{Ancho}(\text{mH}) \wedge \text{Alto}(\text{mE}) = \text{Alto}(\text{mH})$
10. $\wedge (\forall i: \text{nat})(\forall j: \text{nat}) 0 < i < \text{Ancho}(\text{mAg}) \wedge 0 < j < \text{Alto}(\text{mAg}) \Rightarrow_L (\pi_1(\text{Valor}(\text{mAg}, i, j)) \iff (\exists p: \text{placa}) (\text{def?}(p, e.\text{agentes}) \wedge_L (\text{obtener}(p, e.\text{agentes}).\text{pos} = \langle i, j \rangle \wedge \text{esPermutacion}(\text{SecuSuby}(\pi_2(\text{Valor}(\text{mAg}, i, j))), \text{secuenciarDic}(e.\text{agentes})) \wedge \pi_1(\text{Siguiente}(\pi_2(\text{Valor}(\text{mAg}, i, j)))) = \text{obtener}(p, e.\text{agentes}).\text{pl}))$
11. $\wedge (\forall i: \text{nat})(\forall j: \text{nat}) 0 < i < \text{Ancho}(\text{mH}) \wedge 0 < j < \text{Alto}(\text{mH}) \Rightarrow_L (\pi_1(\text{Valor}(\text{mH}, i, j)) \iff (\exists n: \text{nombre})(\text{def?}(n, e.\text{hippies}) \wedge_L \text{obtener}(n, e.\text{hippies}) = \langle i, j \rangle \wedge \pi_1(\text{Siguiente}(\pi_2(\text{Valor}(\text{mH}, i, j)))) = n \wedge \text{esPermutacion}(\text{SecuSuby}(\pi_2(\text{Valor}(\text{mH}, i, j))), \text{secuenciarDic}(e.\text{hippies})))$
12. $\wedge (\forall i: \text{nat})(\forall j: \text{nat}) 0 < i < \text{Ancho}(\text{mE}) \wedge 0 < j < \text{Alto}(\text{mE}) \Rightarrow_L (\pi_1(\text{Valor}(\text{mE}, i, j)) \iff (\exists n: \text{nombre})(\text{def?}(n, e.\text{estudiantes}) \wedge_L \text{obtener}(n, e.\text{estudiantes}) = \langle i, j \rangle \wedge \pi_1(\text{Siguiente}(\pi_2(\text{Valor}(\text{mE}, i, j)))) = n \wedge \text{esPermutacion}(\text{SecuSuby}(\pi_2(\text{Valor}(\text{mE}, i, j))), \text{secuenciarDic}(e.\text{estudiantes})))$
13. $\# \text{claves}(e.\text{agentes}) \geq 1 \Rightarrow (\exists p: \text{placa}) \text{def?}(p, e.\text{agentes}) \wedge_L (\text{obtener}(p, e.\text{agentes}).\text{pl} = e.\text{masVigilante}.pl \wedge \text{obtener}(p, e.\text{agentes}).\text{premios} = e.\text{masVigilante}.premios \wedge ((\forall p': \text{placa}) \text{def?}(p', e.\text{agentes}) \wedge p \neq p' \Rightarrow_L (\text{obtener}(p, e.\text{agentes}).\text{premios} \geq \text{obtener}(p', e.\text{agentes}).\text{premios} \wedge (\text{obtener}(p, e.\text{agentes}).\text{premios} = \text{obtener}(p', e.\text{agentes}).\text{premios} \Rightarrow p < p'))))$


```

armarMultiPos : estr  $\longrightarrow$  multiconj(posicion)
armarMultiPos(e)  $\equiv$  multiSignificados(soloPosAgentes(e.agentes,claves(e.agentes)),claves(e.placasPos))  $\cup$ 
                    multiSignificados(e.hippies,claves(e.hippies))  $\cup$ 
                    multiSignificados(e.estudiantes,claves(e.estudiantes))

soloPosAgentes : dicc(nat  $\times$  datosAg) d  $\times$  conj( $\beta$ ) c  $\longrightarrow$  dicc(nat,posicion) {c  $\subseteq$  claves(d)}
soloPosAgentes(d,c)  $\equiv$  if  $\emptyset?(c)$  then
    Vacio()
    else
        definir(dameUno(c), obtener(dameUno(c),d).pos,soloPosAgentes(d,sinUno(c)))
    fi

multiSignificados : dicc( $\beta \times \alpha$ ) d  $\times$  conj( $\beta$ ) c  $\longrightarrow$  multiconj( $\alpha$ ) { $(\forall cl : \beta) cl \in c \Rightarrow \text{def?}(cl,d)$ }
multiSignificados(d,c)  $\equiv$  if  $\emptyset?(c)$  then  $\emptyset$  else Ag(obtener(dameUno(c),d),multiSignificados(d,sinUno(c))) fi

unionSignificados : dicc( $\beta \times \alpha$ ) d  $\longrightarrow$  conj( $\alpha$ )
unionSignificados(d)  $\equiv$  multiAConj(multiSignificados(d,claves(d)))

multiAConj : multiconj( $\alpha$ )  $\longrightarrow$  conj( $\alpha$ )
multiAConj(mc)  $\equiv$  if mc =  $\emptyset$  then  $\emptyset$  else Ag(dameUno(mc),multiAConj(sinUno(mc))) fi

secuenciarDic : dicc( $\beta \times \alpha$ )  $\longrightarrow$  secu( $\langle \beta, \alpha \rangle$ )
secuenciarDic(d)  $\equiv$  secuenciarDicAux(d,claves(d))

secuenciarDicAux : dicc( $\beta \times \alpha$ ) d  $\times$  conj( $\beta$ ) c  $\longrightarrow$  secu( $\langle \beta, \alpha \rangle$ ) { $(\forall cl : \beta) cl \in c \Rightarrow \text{def?}(cl,d)$ }
secuenciarDicAux(d,c)  $\equiv$  if  $\emptyset?(c)$  then
     $\langle \rangle$ 
    else
         $\langle \text{dameUno}(c), \text{obtener}(\text{dameUno}(c),d) \rangle \bullet \text{secuenciarDicAux}(d, \text{sinUno}(c))$ 
    fi

ordenar $\pi_1$  : secu( $\langle \beta \times \alpha \rangle$ )  $\longrightarrow$  secu( $\langle \beta, \alpha \rangle$ )
ordenar $\pi_1(s)$   $\equiv$  if vacia?(s) then  $\langle \rangle$  else min $\pi_1(s) \bullet \text{ordenar}\pi_1(\text{sacar}(\text{min}\pi_1(s),s))$  fi

min $\pi_1$  : secu( $\langle \beta \times \alpha \rangle$ ) s  $\longrightarrow$   $\langle \beta, \alpha \rangle$  { $\neg \text{vacía?}(s)$ }
min $\pi_1(s)$   $\equiv$  minAux $\pi_1(s, \text{prim}(s))$ 

minAux $\pi_1$  : secu( $\langle \beta \times \alpha \rangle$ ) s  $\times$   $\langle \beta \times \alpha \rangle$   $\longrightarrow$   $\langle \beta, \alpha \rangle$  { $\neg \text{vacía?}(s)$ }
minAux $\pi_1(s,e)$   $\equiv$  if long(s) = 1 then
    if  $\pi_1(\text{prim}(s)) < \pi_1(e)$  then prim(s) else e fi
    else
        minAux $\pi_1(\text{fin}(s), \text{if } \pi_1(\text{prim}(s)) < \pi_1(e) \text{ then } \text{prim}(s) \text{ else } e)$  fi
    fi

sacar :  $\alpha \times \text{secu}(\alpha)$  s  $\longrightarrow$  secu( $\alpha$ ) {esta?(a,s)}
sacar(a,s)  $\equiv$  if prim(s) = a then fin(s) else prim(s)  $\bullet$  sacar(a,fin(s)) fi

Abs : estr e  $\longrightarrow$  campusSeg {Rep(e)}
Abs(e)  $\equiv$  cs: campusSeg / e.cp = campus(cs)  $\wedge$  claves(e.estudiantes) = estudiantes(cs)
     $\wedge$  claves(e.hippies) = hippies(cs)  $\wedge$  claves(e.agentes) = agentes(cs)  $\wedge$ 
    ( $\forall a$ : agente)((a  $\in$  agentes(cs)  $\iff$  def?(a,e.agentes))  $\wedge_L$  (a  $\in$  agentes(cs)  $\Rightarrow_L$ 
    (obtener(a,e.agentes).sanciones = cantSanciones(a,cs)  $\wedge$ 
    obtener(a,e.agentes).premios = cantHippiesAtrapados(a,cs)  $\wedge$ 
    obtener(a,e.agentes).posicion = posAgente(a,cs))))  $\wedge_L$ 
    ( $\forall n$ : nombre)(def?(n,e.hippies)  $\Rightarrow_L$  posEstudianteYHippie(n,cs) = obtener(n,e.hippies))  $\wedge$ 
    (def?(n,e.estudiantes)  $\Rightarrow_L$  posEstudianteYHippie(n,cs) = obtener(n,e.estudiantes))

```

Algoritmos

Algorithm 1 Campus

```
1: procedure ICAMPUS(in c : estr)  $\rightarrow res : campus$ 
2:    $res \leftarrow c.cp$   $\triangleright \mathcal{O}(1)$ 
```

Complejidad: $\mathcal{O}(1)$

Algorithm 2 Estudiantes

```
1: procedure IESTUDIANTES(in c : estr)  $\rightarrow res : itClavesDiccN(string)$ 
2:    $res \leftarrow \text{CrearIt}(c.estudiantes)$   $\triangleright \mathcal{O}(1)$ 
```

Complejidad: $\mathcal{O}(1)$

Algorithm 3 Hippies

```
1: procedure IHIPPIES(in c : estr)  $\rightarrow res : itClavesDiccN(string)$ 
2:    $res \leftarrow \text{CrearIt}(c.hippies)$   $\triangleright \mathcal{O}(1)$ 
```

Complejidad: $\mathcal{O}(1)$

Algorithm 4 Agentes

```
1: procedure IAGENTES(in c : estr)  $\rightarrow res : itClavesDiccLog(nat)$ 
2:    $res \leftarrow \text{CrearIt}(c.agentes)$   $\triangleright \mathcal{O}(1)$ 
```

Complejidad: $\mathcal{O}(1)$

Algorithm 5 Posicion de Estudiante Y Hippie

```
1: procedure IPOSESTUDIANTEYHIPPIE(in c : estr, in n : nombre)  $\rightarrow res : posicion$ 
2:   if Definido(c.hippies, n) then  $\triangleright \mathcal{O}(|h_m|)$ 
3:      $res \leftarrow \text{Obtener}(n, c.hippies)$   $\triangleright \mathcal{O}(|h_m|)$ 
4:   else
5:     if Definido(c.estudiantes, n) then  $\triangleright \mathcal{O}(|e_m|)$ 
6:        $res \leftarrow \text{Obtener}(n, c.estudiantes)$   $\triangleright \mathcal{O}(|e_m|)$ 
```

Complejidad: $\mathcal{O}(|n_m|)$

Justificación: h_m y e_m son el hippie y estudiante con nombre mas largo, sea cual sea la rama del if a la que se entra la complejidad llega a ser $\mathcal{O}(|n_m|) + \mathcal{O}(|n_m|) = \mathcal{O}(|n_m|)$ con $|n_m| = \max(|h_m|, |e_m|)$

Algorithm 6 Posicion de Agente

```
1: procedure IPOSAGENTE(in c : estr, in p : placa)  $\rightarrow res : posicion$ 
2:    $res \leftarrow \text{Obtener}(c.agentes, pl)$   $\triangleright \mathcal{O}(1)$  promedio
```

Complejidad: $\mathcal{O}(1)$ promedio

Algorithm 7 Cantidad de Sanciones

```
1: procedure ICANTSANCIONES(in c : estr, in pl : placa)  $\rightarrow res : nat$ 
2:    $res \leftarrow \text{Obtener}(c.agentes, pl).sanciones$   $\triangleright \mathcal{O}(1)$  promedio
```

Complejidad: $\mathcal{O}(1)$ promedio

Algorithm 8 Cantidad de Hippies Atrapados

```

1: procedure ICANTHIPPIESATRAPADOS(in c : estr, in pl : placa)  $\rightarrow res : nat$ 
2:    $res \leftarrow \text{Obtener}(c.agentes, pl).premios$   $\triangleright \mathcal{O}(1)$  promedio

```

Complejidad: $\mathcal{O}(1)$ promedio

Algorithm 9 Comenzar Rastrillaje

```

1: procedure ICOMENZARRASTRILLAJE(in c : campus, in d : diccLineal(placa, posicion))  $\rightarrow res : estr$ 
2:    $res.cp \leftarrow \text{Copiar}(c)$   $\triangleright \mathcal{O}(h * a)$ 
3:    $res.posiciones.mAg \leftarrow \text{CrearMatriz}(\text{Filas}(c), \text{Columnas}(c), \text{CrearItDiccLog}())$   $\triangleright \mathcal{O}(h * a)$ 
4:    $res.posiciones.mH \leftarrow \text{CrearMatriz}(\text{Filas}(c), \text{Columnas}(c), \text{CrearItDiccN}())$   $\triangleright \mathcal{O}(h * a)$ 
5:    $res.posiciones.mE \leftarrow \text{CrearMatriz}(\text{Filas}(c), \text{Columnas}(c), \text{CrearItDiccN}())$   $\triangleright \mathcal{O}(h * a)$ 
6:    $res.hippies \leftarrow \text{Vacio}()$   $\triangleright \mathcal{O}(1)$ 
7:    $res.estudiantes \leftarrow \text{Vacio}()$   $\triangleright \mathcal{O}(1)$ 
8:    $res.sanciones.lista \leftarrow \text{Vacía}()$   $\triangleright \mathcal{O}(1)$ 
9:    $res.sanciones.rapidas \leftarrow \text{Vacio}()$   $\triangleright \mathcal{O}(1)$ 
10:   $res.sanciones.actualizar \leftarrow \text{true}$   $\triangleright \mathcal{O}(1)$ 
11:   $res.agentesRapido \leftarrow \text{Vacio}(\#Claves(d))$   $\triangleright \mathcal{O}(1)$ 
12:   $res.agentes \leftarrow \text{Vacio}(\#Claves(d))$   $\triangleright \mathcal{O}(1)$ 
13:   $itDicc(placa, posicion) \leftarrow \text{CrearIt}(d)$   $\triangleright \mathcal{O}(1)$ 
14:  if HaySiguiente(it) then  $\triangleright \mathcal{O}(1)$ 
15:     $conj(placa) \leftarrow \text{Vacio}()$   $\triangleright \mathcal{O}(1)$ 
16:     $itLista((nat, conj(placa))) \leftarrow \text{AgregarAtras}(res.sanciones.lista, cp)$   $\triangleright (\text{un conjunto vacio}) \mathcal{O}(1)$ 
17:     $bool \ f \leftarrow \text{true}$   $\triangleright \mathcal{O}(1)$ 
18:    while HaySiguiente(it) do  $\triangleright n * \mathcal{O}(1)$ 
19:       $placa \ pl \leftarrow \text{Copia}(\text{SiguienteClave}(it))$   $\triangleright \mathcal{O}(1)$ 
20:       $posicion \ pos \leftarrow \text{CrearPosicion}(X(\text{SiguienteSignificado}(it)), Y(\text{SiguienteSignificado}(it)))$   $\triangleright \mathcal{O}(1)$ 
21:       $itConj(placa) \ itc \leftarrow \text{AgregarRapido}(\text{Siguiente}(itlist).s, pl)$   $\triangleright \mathcal{O}(1)$ 
22:       $datosAg \ ag \leftarrow \langle pl, 0, 0, pos, itlist, itc \rangle$   $\triangleright \mathcal{O}(1)$ 
23:       $itDicLog(nat, datosAg) \ itdl \leftarrow \text{DefinirLento}(res.agentes, pl, ag)$   $\triangleright \mathcal{O}(n)$ 
24:       $res.posiciones.mAg[X(pos)][Y(pos)] \leftarrow itdl$   $\triangleright \mathcal{O}(1)$ 
25:       $\text{Definir}(res.agentesRapido, pl, itdl)$   $\triangleright \mathcal{O}(1)$  promedio
26:      if ( $f \parallel res.masVigilante.placa \geq pl$ ) then  $\triangleright \mathcal{O}(1)$ 
27:         $res.masVigilante.pl \leftarrow ag.pl$   $\triangleright \mathcal{O}(1)$ 
28:         $res.masVigilante.premios \leftarrow ag.premios$   $\triangleright \mathcal{O}(1)$ 
29:         $f \leftarrow \text{false}$   $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(h * a + n^2) + \mathcal{O}(n)$ prom

Donde

- h es $\text{Filas}(c)$
- a es $\text{Columnas}(c)$
- n es $\#Claves(d)$

Justificación: Sumo los $\mathcal{O}(1)$ continuos para aclarar un poco ($C * \mathcal{O}(1) = \mathcal{O}(1)$)

$4 * \mathcal{O}(h * a) + \mathcal{O}(1) + n * \mathcal{O}(1) * (\mathcal{O}(n) + \mathcal{O}(1) \text{ prom}) = \mathcal{O}(h * a) + \mathcal{O}(n) * (\mathcal{O}(n) + \mathcal{O}(1) \text{ prom}) = \mathcal{O}(h * a) + \mathcal{O}(n^2) + \mathcal{O}(n)$
 $\text{prom} = \mathcal{O}(h * a + n^2) + \mathcal{O}(n) \text{ prom}$

Algorithm 10 Ingresar Estudiante

```

1: procedure IINGRESARESTUDIANTE(in/out  $c$  : estr, in  $n$  : nombre, in  $p$  : posicion)
2:    $c.pos.mE[X(p)][Y(p)] \leftarrow \langle \text{true}, \text{Definir}(c.estudiantes, n, p) \rangle$   $\triangleright \mathcal{O}(|e_m|)$ 
3:    $enhippizar(c, p)$   $\triangleright \mathcal{O}(|h_m|)$ 
4:    $estudiantizar(c, p)$   $\triangleright \mathcal{O}(|e_m|)$ 
5:    $premiar(c, p)$   $\triangleright \mathcal{O}(1)$ 
6:    $capturar(c, p)$   $\triangleright \mathcal{O}(|h_m|)$ 
7:    $sancionar(c, p)$   $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(|n_m|)$ Justificacion: $\mathcal{O}(|e_m|) + \mathcal{O}(|h_m|) + \mathcal{O}(|e_m|) + \mathcal{O}(|h_m|) = \mathcal{O}(2|e_m|) + \mathcal{O}(2|h_m|) = \mathcal{O}(2\max(|e_m|, |h_m|)) = \mathcal{O}(|n_m|)$ **Algorithm 11** Ingresar Hippie

```

1: procedure IINGRESARHIPPIE(in/out  $c$  : estr, in  $n$  : nombre, in  $p$  : posicion)
2:    $c.pos.mH[X(p)][Y(p)] \leftarrow \langle \text{true}, \text{Definir}(c.hippies, n, p) \rangle$   $\triangleright \mathcal{O}(1)$ 
3:    $enhippizar(c, p)$   $\triangleright \mathcal{O}(|h_m|)$ 
4:    $premiar(c, p)$   $\triangleright \mathcal{O}(1)$ 
5:    $capturar(c, p)$   $\triangleright \mathcal{O}(1)$ 
6:    $sancionar(c, p)$   $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(|n_m|)$ Justificacion: $\mathcal{O}(|h_m|) = \mathcal{O}(|n_m|)$ **Algorithm 12** Mover Estudiante

```

1: procedure IMOVERESTUDIANTE(in/out  $c$  : estr, in  $n$  : nombre, in  $d$  : direccion)
2:    $posicion\ p \leftarrow \text{PosEstudianteYHippie}(c, n)$   $\triangleright \mathcal{O}(|n_m|)$ 
3:    $c.pos.mE[X(p)][Y(p)].def \leftarrow \text{false}$   $\triangleright \mathcal{O}(1)$ 
4:    $posicion\ proxp \leftarrow \text{ProxPosicion}(c.cp, p, d)$   $\triangleright \mathcal{O}(1)$ 
5:   if EsIngreso( $c.cp, proxp$ ) then  $\triangleright \mathcal{O}(1)$ 
6:      $\text{EliminarSiguiente}(c.pos.mE[X(p)][Y(p)].dato)$   $\triangleright \mathcal{O}(|e_m|)$ 
7:   else
8:      $c.pos.mE[X(proxp)][Y(proxp)].def \leftarrow \text{true}$   $\triangleright \mathcal{O}(1)$ 
9:      $c.pos.mE[X(proxp)][Y(proxp)].dato \leftarrow c.pos.mE[X(p)][Y(p)].dato$   $\triangleright \mathcal{O}(1)$ 
10:     $enhippizar(c, p)$   $\triangleright \mathcal{O}(|h_m|)$ 
11:     $estudiantizar(c, p)$   $\triangleright \mathcal{O}(|e_m|)$ 
12:     $premiar(c, p)$   $\triangleright \mathcal{O}(1)$ 
13:     $capturar(c, p)$   $\triangleright \mathcal{O}(|h_m|)$ 
14:     $sancionar(c, p)$   $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(|n_m|)$ Justificacion: $\mathcal{O}(|n_m|) + \mathcal{O}(|h_m|) + \mathcal{O}(|e_m|) + \mathcal{O}(|h_m|) = \mathcal{O}(|n_m|) + \mathcal{O}(2|h_m|) + \mathcal{O}(|e_m|) = \mathcal{O}(|n_m|) + \mathcal{O}(|e_m| + |h_m|)$
 $= \mathcal{O}(|n_m|) + \mathcal{O}(|n_m|) = \mathcal{O}(|n_m|)$

Algorithm 13 Mover Hippie

```

1: procedure IMOVERHIPPIE(in/out c : estr, in n : nombre)
2:   posicion p ← PosEstudianteYHippie(c, n)                                ▷  $\mathcal{O}(|n_m|)$ 
3:   c.pos.mH[X(p)] [Y(p)].def ← false                                    ▷  $\mathcal{O}(1)$ 
4:   posicion proxp ← nuevaPosHippieOAgente(c, p, CrearIt(c.estudiantes))    ▷  $\mathcal{O}(N_e)$ 
5:   c.pos.mH[X(proxp)] [Y(proxp)].def ← true                            ▷  $\mathcal{O}(1)$ 
6:   c.pos.mH[X(proxp)] [Y(proxp)].dato ← c.pos.mH[X(p)] [Y(p)].dato    ▷  $\mathcal{O}(1)$ 
7:   enhippizar(c, p)                                                    ▷  $\mathcal{O}(|h_m|)$ 
8:   premiar(c, p)                                                        ▷  $\mathcal{O}(1)$ 
9:   capturar(c, p)                                                      ▷  $\mathcal{O}(|h_m|)$ 
10:  sancionar(c, p)                                                      ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(|n_m|) + \mathcal{O}(N_e)$

Justificacion: $\mathcal{O}(|n_m|) + \mathcal{O}(N_e) + \mathcal{O}(|h_m|) + \mathcal{O}(|h_m|) = \mathcal{O}(n_m) + \mathcal{O}(N_e) + 2\mathcal{O}(|h_m|) = \mathcal{O}(|n_m|) + \mathcal{O}(N_e) + 2\mathcal{O}(|n_m|)$
 $= 3\mathcal{O}(|n_m|) + \mathcal{O}(N_e) = \mathcal{O}(|n_m|) + \mathcal{O}(N_e)$

Algorithm 14 Mover Agente

```

1: procedure IMOVERAGENTE(in/out c : estr, in pl : placa)
2:   posicion p ← Obtener(c.agentes, pl)                                ▷  $\mathcal{O}(\log N_a)$ 
3:   c.pos.mAg[X(p)] [Y(p)].def ← false                                    ▷  $\mathcal{O}(1)$ 
4:   posicion proxp ← nuevaPosHippieOAgente(c, p, CrearIt(c.hippies))    ▷  $\mathcal{O}(N_h)$ 
5:   c.pos.mAg[X(proxp)] [Y(proxp)].def ← true                            ▷  $\mathcal{O}(1)$ 
6:   c.pos.mAg[X(proxp)] [Y(proxp)].dato ← c.pos.mAg[X(p)] [Y(p)].dato    ▷  $\mathcal{O}(1)$ 
7:   premiar(c, p)                                                        ▷  $\mathcal{O}(1)$ 
8:   capturar(c, p)                                                      ▷  $\mathcal{O}(|h_m|)$ 
9:   sancionar(c, p)                                                      ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(|n_m|) + \mathcal{O}(\log N_a) + \mathcal{O}(N_h)$ **Algorithm 15** Con K Sanciones

```

1: procedure ICONKSANCIONES(in/out c : estr, in k : nat) → res:conj(placa)
2:   if c.sanciones.actualizar then                                       ▷  $\mathcal{O}(1)$ 
3:     actualizarSanciones(c)                                             ▷  $\mathcal{O}(N_a)$ 
4:   if Def(c.sanciones, k) then                                           ▷  $\mathcal{O}(\log N_a)$ 
5:     res ← Obtener(c.sanciones.rapidas, k)                             ▷  $\mathcal{O}(\log N_a)$ 
6:   else
7:     res ← Vacio()                                                       ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(N_a)$ si hubo sanciones entre la ultima con K sanciones o $\mathcal{O}(\log N_a)$ si no.**Algorithm 16** Con Mismas Sanciones

```

1: procedure ICONMISMASANCIONES(in c : estr, in pl : placa) → res:conj(placa)
2:   res ← Siguiente(Obtener(c.agentesRapido, pl).kSanc)                ▷  $\mathcal{O}(1)$  promedio

```

Complejidad: $\mathcal{O}(1)$ promedio**Algorithm 17** Cantidad de Hippies

```

1: procedure ICANTHIPPIES(in c : estr) → res:nat
2:   res ← #Claves(c.hippies)                                             ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$

Algorithm 18 Cantidad de Estudiantes

```

1: procedure ICANTESTUDIANTES(in c : estr)  $\rightarrow res: nat$ 
2:    $res \leftarrow \#Claves(c.estudiantes)$   $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ **Algorithm 19** Mas Vigilante

```

1: procedure IMASVIGILANTE(in c : estr)  $\rightarrow res: placa$ 
2:    $res \leftarrow c.masVigilante.pl$   $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ **Algorithm 20** Actualizar Sanciones

```

1: procedure IACTUALIZARSANCIONES(in/out c : estr)
2:    $it \leftarrow \text{CrearIt}(e.sanciones.lista)$   $\triangleright \mathcal{O}(1)$ 
3:   while HaySiguiente(it) do  $\triangleright n * \mathcal{O}(1)$ 
4:     Definir(e.sanciones.rapidas, Siguiente(it).c, Siguiente(it).s)  $\triangleright \mathcal{O}(\#(Claves(Siguiente(it).s)))$ 
5:     Avanzar(it)  $\triangleright \mathcal{O}(1)$ 
6:    $c.sanciones.actualizar \leftarrow false$   $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(n)$ Justificacion: Estableciendo n como la cantidad de sanciones diferentes. Si bien el definir de diccionario logaritmico funciona por copia, la sumatoria de todos la conjuntos de la lista de sanciones es menor o igual n (en peor caso n). Por lo tanto la complejidad es, en todos los casos $\mathcal{O}(2n) = \mathcal{O}(n)$

Algorithm 21 Nueva Posicion Hippie o Agente

```

1: procedure NUEVAPOS HIPPIEOAGENTE(in c : estr, in p : posicion, in/out itObj :
   itDiccN(string,posicion))  $\rightarrow res : posicion$ 
2:    $nat\ min \leftarrow 0$   $\triangleright \mathcal{O}(1)$ 
3:    $posicion\ objetivo$   $\triangleright \mathcal{O}(1)$ 
4:   while HaySiguiente(itObj) do  $\triangleright n * \mathcal{O}(1)$ 
5:     if  $min == 0 \parallel min \geq \text{DistanciaPosiciones}(p, \text{SiguienteSignificado}(itObj))$  then  $\triangleright \mathcal{O}(1)$ 
6:        $min \leftarrow \text{DistanciaPosiciones}(p, \text{SiguienteSignificado}(itObj))$   $\triangleright \mathcal{O}(1)$ 
7:        $objetivo \leftarrow \text{SiguienteSignificado}(itObj)$   $\triangleright \mathcal{O}(1)$ 
8:       Avanzar(itObj)  $\triangleright \mathcal{O}(1)$ 
9:   if  $min == 0$  then  $\triangleright \mathcal{O}(1)$ 
10:     $objetivo \leftarrow \text{Siguiente}(\text{CrearIt}(\text{IngresosMasCercanos}(c.cp, p)))$   $\triangleright \mathcal{O}(1)$ 
11:     $conj(posicion)\ vecinosLibres \leftarrow \text{libres}(c, \text{Vecinos}(c.cp, p))$   $\triangleright \mathcal{O}(k)$ 
12:     $itconj(posicion)\ itVL \leftarrow \text{CrearIt}(vecinosLibres)$   $\triangleright \mathcal{O}(1)$ 
13:     $min \leftarrow 0$   $\triangleright \mathcal{O}(1)$ 
14:    while HaySiguiente(itVL) do  $\triangleright m * \mathcal{O}(1)$ 
15:      if  $min == 0 \parallel min \geq \text{DistanciaPosiciones}(objetivo, \text{Siguiente}(itVL))$  then
16:         $min \leftarrow \text{DistanciaPosiciones}(objetivo, \text{Siguiente}(itVL))$   $\triangleright \mathcal{O}(1)$ 
17:         $res \leftarrow \text{Siguiente}(itVL)$   $\triangleright \mathcal{O}(1)$ 
18:      Avanzar(itVL)  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(n)$ siendo n la cantidad de iteraciones de *itObj*Justificacion: $n * \mathcal{O}(1) + \mathcal{O}(k) + \mathcal{O}(m) = \mathcal{O}(n + k + m)$

Donde

- n es la cantidad de iteraciones de *itObj*
- m es la cantidad de posiciones libres de vecinos de una posicion, acotado por 4
- k es la cantidad de vecinos de una posicion, acotado por 4

$$\mathcal{O}(n + 4 + 4) = \mathcal{O}(n + 8) = \mathcal{O}(n)$$

Algorithm 22 Posiciones Libres

```

1: procedure LIBRES(in c : estr, in cp : conj(posicion))  $\rightarrow res : conj(posicion)$ 
2:   itconj(posicion) itc  $\leftarrow$  CrearIt(cp)  $\triangleright \mathcal{O}(1)$ 
3:   res  $\leftarrow$  Vacio()  $\triangleright \mathcal{O}(1)$ 
4:   while HaySiguiente(itc) do  $\triangleright n * \mathcal{O}(1)$ 
5:     if !ocupada(c, Siguiente(p)) then  $\triangleright \mathcal{O}(1)$ 
6:       AgregarRapido(res, Siguiente(p))  $\triangleright \mathcal{O}(1)$ 
7:       Avanzar(itc)  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(n)$ siendo $n \# \text{Claves}(cp)$

Algorithm 23 De estudiante a hippie

```

1: procedure ENHIPPIZAR(in/out c : estr, in p : posicion)
2:   itConj(posicion) itc  $\leftarrow$  CrearIt(AgregarRapido(p, Vecinos(c.cp, p)))  $\triangleright \mathcal{O}(1)$ 
3:   while HaySiguiente(itc) do  $\triangleright n + \mathcal{O}(1)$ 
4:     posicion p  $\leftarrow$  Siguiente(itc)  $\triangleright \mathcal{O}(1)$ 
5:     if esEstudiante(c, p) && enhippizado(c, p) then  $\triangleright \mathcal{O}(1)$ 
6:       itDiccN(string, posicion) itEaH  $\leftarrow$  c.pos.mE[X(p)][Y(p)].dato  $\triangleright \mathcal{O}(1)$ 
7:       EliminarSiguiente(itEaH)  $\triangleright \mathcal{O}(h_m)$ 
8:       c.pos.mE[X(p)][Y(p)].def  $\leftarrow$  false  $\triangleright \mathcal{O}(1)$ 
9:       c.pos.mH[X(p)][Y(p)].def  $\leftarrow$  true  $\triangleright \mathcal{O}(1)$ 
10:      c.pos.mH[X(p)][Y(p)].dato  $\leftarrow$  Definir(c.hippies, n, p)  $\triangleright \mathcal{O}(|h_m|)$ 
11:      Avanzar(itc)  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(|h_m|)$

Justificacion: $\mathcal{O}(1) + (n * \mathcal{O}(1)) * 2\mathcal{O}(|h_m|) = \mathcal{O}(n) * 2\mathcal{O}(|h_m|)$ con n siendo la cantidad de vecinos + 1, acotado por 5.
 $\mathcal{O}(5) * 2\mathcal{O}(|h_m|) = \mathcal{O}(|h_m|)$

Algorithm 24 De hippie a estudiante

```

1: procedure ESTUDIANTIZAR(in/out c : estr, in p : posicion)
2:   itConj(posicion) itc  $\leftarrow$  CrearIt(AgregarRapido(p, Vecinos(c.cp, p)))  $\triangleright \mathcal{O}(1)$ 
3:   while HaySiguiente(itc) do  $\triangleright n + \mathcal{O}(1)$ 
4:     posicion p  $\leftarrow$  Siguiente(itc)  $\triangleright \mathcal{O}(1)$ 
5:     if esHippie(c, p) && estudiantizado(c, p) then  $\triangleright \mathcal{O}(1)$ 
6:       itDiccN(string, posicion) itHaE  $\leftarrow$  c.pos.mH[X(p)][Y(p)]  $\triangleright \mathcal{O}(1)$ 
7:       EliminarSiguiente(itHaE)  $\triangleright \mathcal{O}(|e_m|)$ 
8:       c.pos.mH[X(p)][Y(p)].def  $\leftarrow$  false  $\triangleright \mathcal{O}(1)$ 
9:       c.pos.mE[X(p)][Y(p)].def  $\leftarrow$  true  $\triangleright \mathcal{O}(1)$ 
10:      c.pos.mE[X(p)][Y(p)].dato  $\leftarrow$  Definir(c.estudiantes, n, p)  $\triangleright \mathcal{O}(|e_m|)$ 

```

Complejidad: $\mathcal{O}(|e_m|)$

Justificacion: $\mathcal{O}(1) + (n * \mathcal{O}(1)) * 2\mathcal{O}(|e_m|) = \mathcal{O}(n) * 2\mathcal{O}(|e_m|)$ con n siendo la cantidad de vecinos + 1, acotado por 5.
 $\mathcal{O}(5) * 2\mathcal{O}(|e_m|) = \mathcal{O}(|e_m|)$

Algorithm 25 Premiar Agentes

```

1: procedure PREMIAR(in/out c : estr, in p : posicion)
2:   if rodeado(c,p) && esHippie(c,p) && #Claves(agentes(c,Vecinos(c.cp,p))) then ▷  $\mathcal{O}(k)$ 
3:     itConj(posicion) itca ← agentes(c,Vecinos(c.cp,p)) ▷  $\mathcal{O}(k)$ 
4:     while HaySiguiente(itca) do ▷  $n * \mathcal{O}(1)$ 
5:       Siguiente(c.pos.mAg[X(p)][Y(p)]).premios ++ ▷  $\mathcal{O}(1)$ 
6:       actualizarMasVigilante(c, pos.mAg[X(p)][Y(p)]) ▷  $\mathcal{O}(1)$ 
7:       Avanzar(itca) ▷  $\mathcal{O}(1)$ 
8:   else
9:     itConj(posicion) itc ← CrearIt(Vecinos(c.cp, p)) ▷  $\mathcal{O}(1)$ 
10:    while HaySiguiente(itc) do ▷  $n * \mathcal{O}(1)$ 
11:      posicion p ← Siguiente(itc) ▷  $\mathcal{O}(1)$ 
12:      if rodeado(p) && esHippie(p) then ▷  $\mathcal{O}(1)$ 
13:        itConj(posicion) itca ← agentes(c,Vecinos(c.cp,p)) ▷  $\mathcal{O}(k)$ 
14:        while HaySiguiente(itca) do ▷  $m * \mathcal{O}(1)$ 
15:          pa ← Siguiente(itca) ▷  $\mathcal{O}(1)$ 
16:          Siguiente(pos.mAg[X(pa)][Y(pa)]).premios ++ ▷  $\mathcal{O}(1)$ 
17:          actualizarMasVigilante(c, pos.mAg[X(pa)][Y(pa)]) ▷  $\mathcal{O}(1)$ 
18:          Avanzar(itca) ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ Justificación: $\mathcal{O}(1) + n * \mathcal{O}(1) * (\mathcal{O}(k) + m\mathcal{O}(1)) = \mathcal{O}(n * (k + m))$

Donde

- *n* es la cantidad de vecinos de una posicion, acotado por 4
- *m* es la cantidad de agentes vecinos de una posicion, acotado por 4
- *k* es la cantidad de vecinos de una posicion, acotado por 4

$$\mathcal{O}(4 * (4 + 4)) = \mathcal{O}(32) = \mathcal{O}(1)$$

Algorithm 26 Capturar Hippies

```

1: procedure CAPTURAR(in/out c : estr, in p : posicion)
2:   if rodeado(c,p) && esHippie(c,p) && #Claves(agentes(c,Vecinos(c.cp,p))) ≥ 1 then ▷  $\mathcal{O}(k)$ 
3:     itDicN(string, posicion) itH ← res.pos.mH[X(p)][Y(p)] ▷  $\mathcal{O}(1)$ 
4:     EliminarSiguiente(itH) ▷  $\mathcal{O}(|h_m|)$ 
5:     res.pos.mE[X(p)][Y(p)].def ← false ▷  $\mathcal{O}(1)$ 
6:   else
7:     itConj(posicion) itc ← CrearIt(Vecinos(c.cp,p)) ▷  $\mathcal{O}(1)$ 
8:     while HaySiguiente(itc) do ▷  $n * \mathcal{O}(1)$ 
9:       posicion p ← Siguiente(itc) ▷  $\mathcal{O}(1)$ 
10:      if rodeado(c,p) && esHippie(c,p) && #Claves(agentes(c,Vecinos(c.cp,p))) ≥ 1 then ▷  $\mathcal{O}(k)$ 
11:        itDicN(string, posicion) itH ← c.pos.mH[X(p)][Y(p)] ▷  $\mathcal{O}(1)$ 
12:        EliminarSiguiente(itH) ▷  $\mathcal{O}(|h_m|)$ 
13:        c.pos.mE[X(p)][Y(p)].def ← false ▷  $\mathcal{O}(1)$ 
14:      Avanzar(itc) ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(|h_m|)$ Justificación: $\mathcal{O}(k) + n * \mathcal{O}(1) * (\mathcal{O}(|h_m|) + \mathcal{O}(k)) = \mathcal{O}(k) + \mathcal{O}(n) * \mathcal{O}(|h_m| + k) = \mathcal{O}(k) + \mathcal{O}(n * |h_m| + n * k) = \mathcal{O}(n * |h_m| + n * k)$ ($n * k > k$)

Donde

- *n* es la cantidad de vecinos de una posicion, acotado por 4
- $|h_m|$ es el hippie con nombre mas largo
- *k* es la cantidad de vecinos de una posicion, acotado por 4

$$\mathcal{O}(4 * |h_m| + 4 * 4) = \mathcal{O}(4 * |h_m| + 16) = \mathcal{O}(4 * |h_m|) = \mathcal{O}(|h_m|)$$

Algorithm 27 Sancionar Agentes

```

1: procedure SANCIONAR(in/out c : estr, in p : posicion)
2:   itConj(posicion) itc ← CrearIt(AgregarRapido(p, Vecinos(c.cp, p))) ▷  $\mathcal{O}(1)$ 
3:   while HaySiguiente(itc) do ▷  $n * \mathcal{O}(1)$ 
4:     posicion p ← Siguiente(itc) ▷  $\mathcal{O}(1)$ 
5:     conj(posicion) cag ← agentes(c, Vecinos(c.cp, p)) ▷  $\mathcal{O}(k)$ 
6:     if rodeado(c, p) && esEstudiante(c, p) && #Claves(cag) ≥ 1 then ▷  $\mathcal{O}(1)$ 
7:       itConj(posicion) itca ← cag ▷  $\mathcal{O}(1)$ 
8:       while HaySiguiente(itca) do ▷  $m * \mathcal{O}(1)$ 
9:         c.sanciones.actualizar ← true ▷  $\mathcal{O}(1)$ 
10:        posicion pa ← Siguiente(itca) ▷  $\mathcal{O}(1)$ 
11:        datosAg datos ← Siguiente(c.pos.mAg[X(pa)]Y(pa)).dato ▷  $\mathcal{O}(1)$ 
12:        datos.sanciones ++
13:        EliminarSiguiente(datos.mismasSanc) ▷  $\mathcal{O}(1)$ 
14:        if #(Siguiente(datos.kSanc).s) == 0 then ▷  $\mathcal{O}(1)$ 
15:          EliminarSiguiente(datos.kSanc) ▷  $\mathcal{O}(1)$ 
16:          if datos.sanciones < SiguienteClave(datos.kSanc) then ▷  $\mathcal{O}(1)$ 
17:            InsertarAdelante(datos.kSanc, datos.sanciones, Vacio()) ▷  $\mathcal{O}(1)$ 
18:          else
19:            Avanzar(datos.kSanc) ▷  $\mathcal{O}(1)$ 
20:            datos.mismasSanc ← AgregarRapido(SiguienteSignificado(datos.kSanc), datos.pl) ▷  $\mathcal{O}(1)$ 
21:            Avanzar(itc) ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ Justificación: $n * (\mathcal{O}(k) + m * \mathcal{O}(1)) = \mathcal{O}(n) * \mathcal{O}(k + m) = \mathcal{O}(n * k + n * m)$

Donde

- *n* es la cantidad de vecinos de una posicion + 1, acotado por 5
- *m* es la cantidad de agentes vecinos de una posicion, acotado por 4
- *k* es la cantidad de vecinos de una posicion, acotado por 4

$$\mathcal{O}(5 * 4 + 5 * 4) = \mathcal{O}(40) = \mathcal{O}(1)$$

Algorithm 28 ¿Se transforma en hippie?

```

1: procedure ENHIPPIZADO(in c : estr, in p : posicion) → res : bool
2:   res ← false ▷  $\mathcal{O}(1)$ 
3:   itConj(posicion) itc ← CrearIt(Vecinos(c.cp, p)) ▷  $\mathcal{O}(1)$ 
4:   nat i ← 0 ▷  $\mathcal{O}(1)$ 
5:   while HaySiguiente(itc) do ▷  $n * \mathcal{O}(1)$ 
6:     if esHippy(c, Siguiente(itc)) then ▷  $\mathcal{O}(1)$ 
7:       i ← i + 1 ▷  $\mathcal{O}(1)$ 
8:       Avanzar(itc) ▷  $\mathcal{O}(1)$ 
9:       res ← (i ≥ 2) ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ Justificación: $\mathcal{O}(1) + n * \mathcal{O}(1) = \mathcal{O}(n)$ siendo *n* la cantidad de vecinos, que como maximo es 4

$$\mathcal{O}(4) = \mathcal{O}(1)$$

Algorithm 29 ¿Es hippie?

```

1: procedure ES_ESTUDIANTE(in c : estr, in p : posicion) → res : bool
2:   res ← c.pos.mH[X(p)]Y(p).def ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$

Algorithm 30 ¿Se transforma en estudiante?

```

1: procedure ENHIPPIZADO(in c : estr, in p : posicion)  $\rightarrow res : bool$ 
2:   res  $\leftarrow$  false  $\triangleright \mathcal{O}(1)$ 
3:   itConj(posicion) itc  $\leftarrow$  CrearIt(Vecinos(c.cp, p))  $\triangleright \mathcal{O}(1)$ 
4:   nat i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
5:   while HaySiguiente(itc) do  $\triangleright n * \mathcal{O}(1)$ 
6:     if esEstudiante(Siguiente(itc)) then  $\triangleright \mathcal{O}(1)$ 
7:       i  $\leftarrow$  i + 1  $\triangleright \mathcal{O}(1)$ 
8:       Avanzar(itc)  $\triangleright \mathcal{O}(1)$ 
       res  $\leftarrow$  (i = 4)  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ Justificacion: $\mathcal{O}(1) + n * \mathcal{O}(1) = \mathcal{O}(n)$ siendo *n* la cantidad de vecinos, que como maximo es 4 $\mathcal{O}(4) = \mathcal{O}(1)$ **Algorithm 31** ¿Es estudiante?

```

1: procedure ESESTUDIANTE(in c : estr, in p : posicion)  $\rightarrow res : bool$ 
2:   res  $\leftarrow$  c.pos.mE[X(p)][Y(p)].def  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ **Algorithm 32** ¿Esta Rodeado?

```

1: procedure RODEADO(in c : estr, in p : posicion)  $\rightarrow res : bool$ 
2:   itConj(posicion) itc  $\leftarrow$  CrearIt(Vecinos(c.cp, p))  $\triangleright \mathcal{O}(1)$ 
3:   nat i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
4:   while HaySiguiente(itc) do  $\triangleright n * \mathcal{O}(1)$ 
5:     if ocupado(Siguiente(p)) then  $\triangleright \mathcal{O}(1)$ 
6:       i ++  $\triangleright \mathcal{O}(1)$ 
       Avanzar(itc)  $\triangleright \mathcal{O}(1)$ 
       res  $\leftarrow$  (i == 4)  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ Justificacion: $n * \mathcal{O}(1) = \mathcal{O}(n)$ siendo *n* la cantidad de vecinos de una posicion, con cota 4 $\mathcal{O}(4) = \mathcal{O}(1)$ **Algorithm 33** ¿Esta Ocupado?

```

1: procedure RODEADO(in c : estr, in p : posicion)  $\rightarrow res : bool$ 
2:   res  $\leftarrow$  (esHippie(c,p) || esAgente(c,p) || esEstudiante(c,p) || Ocupada(c.cp,p))  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ **Algorithm 34** Agentes en posiciones

```

1: procedure AGENTES(in c : estr, in cp : conj(posicion))  $\rightarrow res : conj(posicion)$ 
2:   itConj(posicion) itc  $\leftarrow$  CrearIt(cp)  $\triangleright \mathcal{O}(1)$  res  $\leftarrow$  Vacio();
3:   while HaySiguiente(itc) do  $\triangleright n * \mathcal{O}(1)$ 
4:     if esAgente(Siguiente(p)) then  $\triangleright \mathcal{O}(1)$ 
5:       AgregarRapido(res,p)  $\triangleright \mathcal{O}(1)$ 
       Avanzar(itc)  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(n)$ siendo $n = \#(\text{Claves}(cp))$

Algorithm 35 ¿Es agente?

```
1: procedure ESAGENTE(in c : estr, in p : posicion)  $\rightarrow$  res : bool
2:   res  $\leftarrow$  c.pos.mAg[X(p)][Y(p)].def  $\triangleright \mathcal{O}(1)$ 
```

Complejidad: $\mathcal{O}(1)$

Algorithm 36 Actualizar mas Vigilante

```
1: procedure ACTUALIZARVIGILANTE(in/out c : estr, in datos : datosAg)
2:   if datos.premios > c.masVigilante.premios then  $\triangleright \mathcal{O}(1)$ 
3:     c.masVigilante.pl  $\leftarrow$  datos.pl  $\triangleright \mathcal{O}(1)$ 
4:     c.masVigilante.premios  $\leftarrow$  datos.premios  $\triangleright \mathcal{O}(1)$ 
5:   if datos.premios == c.masVigilante.premios then  $\triangleright \mathcal{O}(1)$ 
6:     if datos.pl < c.masVigilante.pl then  $\triangleright \mathcal{O}(1)$ 
7:       c.masVigilante.pl  $\leftarrow$  datos.pl  $\triangleright \mathcal{O}(1)$ 
8:       c.masVigilante.premios  $\leftarrow$  datos.premios  $\triangleright \mathcal{O}(1)$ 
```

Complejidad: $\mathcal{O}(1)$

2.2. Campus

Interfaz

se explica con: CAMPUS

usa: nat, bool, Matriz(bool), posicion

géneros: campus

Operaciones de Campus

CREARCAMPUS(in ancho: nat, in alto: nat) → res : campus

Pre ≡ {ancho > 0 ∧ alto > 0}

Post ≡ {res =_{obs} crearCampus(ancho, alto)}

Complejidad: $\mathcal{O}(\text{alto} \times \text{ancho})$

Descripción: Crea un campus sin obstáculos.

AGREGAROBSTACULO(in/out c: campus, in p: posicion)

Pre ≡ {posValida?(p, c) ∧ ¬ocupada(p, c) ∧ c = c₀}

Post ≡ {c =_{obs} agregarObstáculo(p, c₀)}

Complejidad: $\mathcal{O}(1)$

Descripción: Agrega un obstáculo en la posición dada.

FILAS(in c: campus) → res : nat

Pre ≡ {true}

Post ≡ {res =_{obs} filas(c)}

Complejidad: $\mathcal{O}(1)$

Descripción: Indica la cantidad de filas del campus dado.

COLUMNAS(in c: campus) → res : nat

Pre ≡ {true}

Post ≡ {res =_{obs} columnas(c)}

Complejidad: $\mathcal{O}(1)$

Descripción: Indica la cantidad de columnas del campus dado.

OCUPADA(in c: campus, in p: posicion) → res : bool

Pre ≡ {posValida?(p, c)}

Post ≡ {res =_{obs} ocupada(p, c)}

Complejidad: $\mathcal{O}(1)$

Descripción: Indica si la posición dada tiene un obstáculo.

POSVALIDA(in c: campus, in p: posicion) → res : bool

Pre ≡ {true}

Post ≡ {res =_{obs} posValida?(p, c)}

Complejidad: $\mathcal{O}(1)$

INGRESOSUPERIOR(in c: campus, in p: posicion) → res : bool

Pre ≡ {true}

Post ≡ {res =_{obs} ingresoSuperior?(p, c)}

Complejidad: $\mathcal{O}(1)$

INGRESOINFERIOR(in c: campus, in p: posicion) → res : bool

Pre ≡ {true}

Post ≡ {res =_{obs} ingresoInferior?(p, c)}

Complejidad: $\mathcal{O}(1)$

ESINGRESO(in c: campus, in p: posicion) → res : bool

Pre ≡ {true}

Post ≡ {res =_{obs} esIngreso?(p, c)}

Complejidad: $\mathcal{O}(1)$

VECINOS(in c : campus, in p : posicion) $\rightarrow res$: conj(posicion)

Pre $\equiv \{posValida?(p,c)\}$

Post $\equiv \{res =_{obs} vecinos?(p,c)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Dada una posición, devuelve las 4 posiciones vecinas.

VECINOSCOMUNES(in c : campus, in p_1 : posicion, in p_2 : posicion) $\rightarrow res$: conj(posicion)

Pre $\equiv \{posValida?(p_1,c) \wedge posValida?(p_2,c)\}$

Post $\equiv \{res =_{obs} vecinosComunes(p_1,p_2,c)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Dadas dos posiciones, devuelve las posiciones que son vecinas de ambas.

VECINOSVALIDOS(in c : campus, in/out cp : conj(posicion))

Pre $\equiv \{c = c_0\}$

Post $\equiv \{c =_{obs} vecinosValidos\}$

Complejidad: $\mathcal{O}(|cp|)$

Descripción: Dado un conjunto de posiciones, devuelve el conjunto resultado de quitar las posiciones no válidas.

PROXPOSICION(in c : campus, in p : posicion, in d : direccion) $\rightarrow res$: posicion

Pre $\equiv \{posValida?(p,c)\}$

Post $\equiv \{res =_{obs} proxPosición(p,d,c)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve el resultado de mover la posición indicada en la dirección dada.

INGRESOSMASCERCANOS(in c : campus, in p : posicion) $\rightarrow res$: conj(posicion)

Pre $\equiv \{posValida?(p,c)\}$

Post $\equiv \{res =_{obs} ingresosMásCercanos(p,c)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve el conjunto de ingresos más cercanos a la posición dada.

COPIAR(in c : campus) $\rightarrow res$: campus

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} c\}$

Complejidad: $\mathcal{O}(Filas(c) * Columnas(c))$

Descripción: Crea una copia del campus.

Representación

Representación del campus

campus se representa con estr

donde estr es Matriz(bool)

$Rep : estr \rightarrow bool$

$Rep(e) \equiv true \iff true$

$Abs : estr\ e \rightarrow campus$

$\{Rep(e)\}$

$Abs(e) \equiv c : campus /$

$filas(c) = Alto(e) \wedge columnas(c) = Ancho(e)$

$\wedge (\forall p : posicion) (posValida?(p,c) \Rightarrow_L ocupada?(p,c) = e[X(p) - 1][Y(p) - 1])$

Algoritmos

Algorithm 37 iCrearCampus

1: **procedure** ICREARCAMPUS(*in* ancho : nat, *in* alto : nat) \rightarrow *res* : estr
 2: *res* \leftarrow Crear(ancho,alto,false) $\triangleright \mathcal{O}(\text{ancho} \times \text{alto} \times \text{copy}(\text{bool}))$

Complejidad: $\mathcal{O}(\text{ancho} * \text{alto})$ Justificación: $\mathcal{O}(\text{copy}(\text{bool})) = \mathcal{O}(1)$

Algorithm 38 iAgregar

1: **procedure** IAGREGAR(*in/out* e : estr, *in* p : posicion)
 2: e[X(p) - 1][Y(p) - 1] \leftarrow true $\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 39 iFila

1: **procedure** IFILA(*in* e : estr) \rightarrow *res* : nat
 2: *res* \leftarrow Alto(e) $\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 40 iColumna

1: **procedure** ICOLUMNA(*in* e : estr) \rightarrow *res* : nat
 2: *res* \leftarrow Ancho(e) $\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 41 iOcupada

1: **procedure** IOcupADA(*in* e : estr, *in* p : posicion) \rightarrow *res* : bool
 2: *res* \leftarrow e[X(p) - 1][Y(p) - 1] $\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 42 iPosValida

procedure IPOSVALIDA(*in* e : estr, *in* p : posicion) \rightarrow *res* : bool
 $0 < X(p) \wedge X(p) \leq \text{Ancho}(e) \wedge 0 < Y(p) \wedge Y(p) \leq \text{Alto}(e)$ $\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 43 iIngresoSuperior

1: **procedure** IINGRESOSUPERIOR(*in* e : estr, *in* p : posicion) \rightarrow *res* : bool
 2: *res* \leftarrow (Y(p) = 1) \wedge iPosValida(e,p) $\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 44 iIngresoInferior

1: **procedure** IINGRESOINFERIOR(*in* e : estr, *in* p : posicion) \rightarrow *res* : bool
 2: *res* \leftarrow (Y(p) = Alto(e)) \wedge iPosValida(e,p) $\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 45 iEsIngreso

```

1: procedure iEsINGRESO(in e : estr, in p : posicion) → res : bool
2:   res ← (iEsIngresoSuperior(e,p) ∨ iEsIngresoInferior(e,p))

```

▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$ **Algorithm 46** iVecinos

```

1: procedure iVECINOS(in e : estr, in p : posicion) → res : conj(posicion)
2:   res ← Vacio() //vacio de conjunto
3:   AgregarRapido(res, <X(p) - 1, Y(p)>)
4:   AgregarRapido(res, <X(p) + 1, Y(p)>)
5:   AgregarRapido(res, <X(p), Y(p) - 1>)
6:   AgregarRapido(res, <X(p), Y(p) + 1>)
7:   res ← iVecinosValidos(e, res)

```

▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(\#res) = \mathcal{O}(4) = \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$ **Algorithm 47** iVecinosComunes

```

1: procedure iVECINOSCOMUNES(in e : estr, in p1 : posicion, in p2 : posicion) → res : conj(posicion)
2:   vec1 ← iVecinos(e, p1)
3:   vec2 ← iVecinos(e, p2)
4:   res ← Vacio() //vacio de conjunto
5:   it1 ← CrearIt(vec1)
6:   while HaySiguiente(it1) do
7:     it2 ← CrearIt(vec2)
8:     while HaySiguiente(it2) do
9:       if Siguiente(it1) = Siguiente(it2) then
10:        AgregarRapido(res, Siguiente(it1))
11:        Avanzar(it2)
12:        Avanzar(it1)

```

▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ Se repite 4 veces y lo de adentro es $\mathcal{O}(1) \Rightarrow \mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ Se repite 4 veces y lo de adentro es $\mathcal{O}(1) \Rightarrow \mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$ **Algorithm 48** iVecinosValidos

```

1: procedure iVECINOSVALIDOS(in e : estr, in/out cp : conj(posicion))
2:   it ← CrearIt(cp)
3:   while HaySiguiente(it) do
4:     if ¬ iPosValida(e, Siguiente(it)) then
5:       EliminarSiguiente(it)
6:     else
7:       Avanzar(it)

```

▷ $\mathcal{O}(1)$
 ▷ Se repite #cp veces y lo de adentro es $\mathcal{O}(1) \Rightarrow \mathcal{O}(\#cp)$
 ▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$
 ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(\#cp)$ **Algorithm 49** iProxPosicion

```

1: procedure iPROXPOSICION(in e : estr, in p : posicion, in dir : direccion) → res : posicion
2:   res ← <X(p) + β(dir = der) - β(dir = izq), Y(p) + β(dir = abajo) - β(dir = arriba)>

```

▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 50 iIngresosMasCercanos

```

1: procedure iINGRESOSMASCERCANOS(in e : estr, in p : posicion)  $\rightarrow$  res : conj(posicion)
2:   res  $\leftarrow$  Vacio() //vacio de conjunto  $\triangleright \mathcal{O}(1)$ 
3:   if Dist(p, <X(p), 1>) < Dist(p, <Y(p), iFilas(e)>) then  $\triangleright \mathcal{O}(1)$ 
4:     AgregarRapido(res, <X(p), 1>)  $\triangleright \mathcal{O}(1)$ 
5:   else
6:     if Dist(p, <X(p), 1>) > Dist(p, <Y(p), iFilas(e)>) then  $\triangleright \mathcal{O}(1)$ 
7:       AgregarRapido(res, <X(p), iFilas(e)>)  $\triangleright \mathcal{O}(1)$ 
8:     else
9:       AgregarRapido(res, <X(p), 1>)  $\triangleright \mathcal{O}(1)$ 
10:      AgregarRapido(res, <X(p), iFilas(e)>)  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$

Algorithm 51 iCopiar

```

1: procedure iCOPIAR(in e : estr)  $\rightarrow$  res : estr
2:   res  $\leftarrow$  Copiar(e)  $\triangleright \mathcal{O}(\text{Copiar}(e))$ 

```

Complejidad: $\mathcal{O}(\text{Ancho}(e) * \text{Alto}(e))$

Justificación: e src Matriz(bool)

\Rightarrow Copiar(e) usa el copiar de matriz

$\Rightarrow \mathcal{O}(\text{Copiar}(e)) = \mathcal{O}(\text{Ancho}(e) * \text{Alto}(e) * \text{copy}(\text{bool}))$

Además $\mathcal{O}(\text{bool}) = \mathcal{O}(1)$

$\Rightarrow \mathcal{O}(\text{Copiar}(e)) = \mathcal{O}(\text{Ancho}(e) * \text{Alto}(e))$

2.3. Diccionario de Nombres

Interfaz

parámetros formales

géneros σ
función $\text{COPIAR}(\text{in } s : \sigma) \rightarrow \text{res} : \sigma$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{\text{res} =_{\text{obs}} a\}$
Complejidad: $\Theta(\text{copy}(s))$
Descripción: función de copia de σ 's

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

usa: nat, bool

géneros: $\text{diccNom}(\text{string}, \sigma)$

Operaciones de $\text{DiccionarioNombres}(\text{string}, \sigma)$

$\text{VACIO}() \rightarrow \text{res} : \text{diccNom}(\text{string}, \sigma)$

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

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

Complejidad: $\mathcal{O}(1)$

Descripción: Crea un diccionario vacio.

$\text{DEFINIR}(\text{in/out } d : \text{diccNom}(\text{string}, \sigma), \text{in } c : \text{string}, \text{in } s : \sigma) \rightarrow \text{res} : \text{itDiccNom}(\text{string}, \sigma)$

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

Post $\equiv \{d =_{\text{obs}} \text{Definir}(c, s, d_0) \wedge \text{haySiguiente}(\text{res}) \wedge_{\text{L}} \text{siguiente}(\text{res}) =_{\text{obs}} \langle c, s \rangle \wedge \text{alias}(\text{esPermutacion}(\text{SecuSuby}(\text{res}), d))\}$

Complejidad: $\mathcal{O}(|c| + \text{copy}(s))$

Descripción: Agrega un elemento al diccionario y devuelve un iterador apuntando a ese elemento.

$\text{DEFINIDO}(\text{in } d : \text{diccNom}(\text{string}, \sigma), \text{in } c : \text{string}) \rightarrow \text{res} : \text{bool}$

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

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

Complejidad: $\mathcal{O}(|c|)$

Descripción: Devuelve verdadero si y sólo si la clave de entrada esta definida en el diccionario.

$\text{BORRAR}(\text{in/out } d : \text{diccNom}(\text{string}, \sigma), \text{in } c : \text{string})$

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

Post $\equiv \{d =_{\text{obs}} \text{borrar}(c, d_0)\}$

Complejidad: $\mathcal{O}(\# \text{claves}(d) + |c|)$

Descripción: Borra el elemento correspondiente a la clave c del diccionario.

$\text{BORRARRAPIDO}(\text{in/out } d : \text{diccNom}(\text{string}, \sigma), \text{in } it : \text{itDiccNom}(\text{string}, \sigma))$

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

Post $\equiv \{d =_{\text{obs}} \text{borrar}(c, d_0)\}$

Complejidad: $\mathcal{O}(|c|)$

Descripción: Borra el elemento correspondiente a la clave c del diccionario. c es la clave correspondiente al siguiente del iterador.

$\text{OBTENER}(\text{in/out } d : \text{diccNom}(\text{string}, \sigma), \text{in } c : \text{string}) \rightarrow \text{res} : \sigma$

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

Post $\equiv \{\text{alias}(\text{res} =_{\text{obs}} \text{obtener}(c, d)) \wedge (\forall c' : \text{string})(c' \in \text{claves}(s) \Rightarrow_{\text{L}} \text{obtener}(c, d) =_{\text{obs}} \text{obtener}(c, d_0))\}$

Complejidad: $\mathcal{O}(|c|)$

Descripción: Devuelve el significado de la clave dada en el diccionario

Aliasing: res es modificable si y solo si σ es modificable

$\# \text{CLAVES}(\text{in } d : \text{diccNom}(\text{string}, \sigma)) \rightarrow \text{res} : \text{nat}$

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

Post $\equiv \{\text{res} =_{\text{obs}} \# \text{claves}(\text{d})\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve la cantidad de claves definidas en el diccionario

Representación

Representacion del DiccionarioNombres(string, σ)

diccNom(string, σ) se representa con estr

donde **estr** es **tupla**(*lineal*: dicc(string, σ) , *trie*: diccT(string, σ))

Donde diccT(string, σ) es Lista(**tupla**<*letra* : char, *significado* : puntero(σ), *prox* : diccT(string, σ)>)

Rep : **estr** \rightarrow bool

Rep(*e*) \equiv true \iff **Rep**(*e*.lineal) \wedge tamaño(*e*.trie) $< 256 \wedge$ SinRepetidos(Letras(*e*.trie))
(\forall dT : diccT(string, σ)) (Esta?(dT, *e*.trie.prox) \Rightarrow_L **Rep**(dT))

Letra : li \rightarrow secu(char)

Letra(l) \equiv **if** vacio?(l) **then** <> **else** prim(l).letra • **Letra**(fin(l)) **fi**

Donde li es Lista(**tupla**<*letra* : char, *significado* : puntero(σ), *prox* : diccT(string, σ)>)

SinRepetidos : secu(α) \rightarrow bool

SinRepetidos(s) \equiv **if** vacio?(s) **then** true **else** \neg esta?(prim(s),fin(s)) \wedge SinRepetidos(fin(s)) **fi**

Abs : **estr** *e* \rightarrow diccNom(string, σ)

{**Rep**(*e*)}

Abs(*e*) \equiv dN : diccNom(string, σ) /

Abs(*e*.lineal) $=_{\text{obs}}$ dN

\wedge (\forall c : string) (Def?(c,d) \iff Definido(c,*e*.trie))

\wedge (\forall c : string) (Def?(c,d) \Rightarrow_L Obtener(c,d) $=_{\text{obs}}$ Significado(c, *e*.trie))

Definido : string \times trie \rightarrow bool

Definido(s,t) \equiv estaLetra(s[0],t) \wedge_L **if** Tamaño(s) == 1 **then** true **else** Definido(c[1:Tamaño(c)-1],t) **fi**

estaLetra : char \times trie \rightarrow bool

estaLetra(c,t) \equiv **if** vacio?(t) **then** false **else** Primero(t).letra $=_{\text{obs}}$ c \vee estaLetra(c, fin(t)) **fi**

Significado : string *s* \times trie *t* \rightarrow σ

{Definido(s,t)}

Significado(s,e) \equiv **if** Tamaño(s) = 1 **then**

* (Buscar(s[0],t).significado)

else

Significado(s[1:Tamaño(s)-1],Buscar(s[0],t).prox)

fi

Buscar : char *c* \times trie *t* \rightarrow piso

{estaLetra(c,t)}

Buscar(c,t) \equiv **if** primero(t).letra = c **then** Primero(t) **else** Buscar(c,fin(t)) **fi**

Donde piso es **tupla**<*letra* : char, *significado* : puntero(σ), *prox* : trie>

Donde trie es diccT(string, σ)

Algoritmos

Algorithm 52 iVacio

procedure iVACIO \rightarrow res : **estr**

res.lineal \leftarrow Vacio() //vacio de diccionario lineal

$\triangleright \mathcal{O}(1)$

res.trie \leftarrow Vacio() //vacio de lista

$\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 53 iDefinir

```

procedure IDEFINIR(in/out e : estr, in c : string, in s :  $\sigma$ )  $\rightarrow$  res : itDiccNom(string,  $\sigma$ )
  res  $\leftarrow$  DefinirRapido(e.lineal, c, s)  $\triangleright \mathcal{O}(\text{copy}(s))$ 
  piso  $\leftarrow$  e.trie  $\triangleright \mathcal{O}(1)$ 
  i  $\leftarrow$  0
  while i < |c|-1 do  $\triangleright$  Se repite |c|-1 veces y lo de adentro es  $\mathcal{O}(1) \Rightarrow \mathcal{O}(|c|-1) = \mathcal{O}(|c|)$ 
    if iEsta(c[i], piso) then
      piso  $\leftarrow$  iBuscar(c[i], piso).prox  $\triangleright \mathcal{O}(1)$ 
    else
      AgregarAdelante(piso, <c[i], NULL, Vacio()>)  $\triangleright \mathcal{O}(1)$ 
      piso  $\leftarrow$  piso[0].prox  $\triangleright \mathcal{O}(1)$ 
    i++  $\triangleright \mathcal{O}(1)$ 
  if iEsta(c[i], piso) then  $\triangleright \mathcal{O}(1)$ 
    iBuscar(c[i], piso).significado  $\leftarrow$  puntero(SiguienteSignificado(res))  $\triangleright \mathcal{O}(1)$ 
  else
    AgregarAdelante(piso, <c[i], puntero(SiguienteSignificado(res)), Vacio()>)  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(|c| + \text{copy}(s))$ **Algorithm 54** iDefinido

```

procedure IDEFINIDO(in e : estr, in c : string)  $\rightarrow$  res : bool
  piso  $\leftarrow$  e.trie  $\triangleright \mathcal{O}(1)$ 
  i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
  while Esta(c[i], piso) do  $\triangleright$  Se repite a lo sumo |c| veces y lo de adentro es  $\mathcal{O}(1) \Rightarrow \mathcal{O}(|c|)$ 
    piso  $\leftarrow$  Buscar(c[i], piso).prox  $\triangleright \mathcal{O}(1)$ 
    i  $\leftarrow$  i++  $\triangleright \mathcal{O}(1)$ 
  res  $\leftarrow$  (i = |c|)

```

Complejidad: $\mathcal{O}(|c|)$ **Algorithm 55** iEsta

ESTA(**in** c : **char**, **in/out** t : *diccT*(string, σ)) \rightarrow res : *bool*

Pre $\equiv \{\text{esta}(c, t)\}$ **Post** $\equiv \{\text{res} =_{\text{obs}} \text{estaLetra}(c, t)\}$ **Complejidad:** $\mathcal{O}(1)$ **Descripción:** Devuelve verdadero si y sólo si el char de entrada esta en alguno de los elementos de la lista.

```

procedure IESTA(in c : char, in t : diccT(string,  $\sigma$ ))  $\rightarrow$  res : bool
  it  $\leftarrow$  crearIr(t)  $\triangleright \mathcal{O}(1)$ 
  while haySiguiente(it)  $\wedge$  Siguiente(it).letra  $\neq$  c do  $\triangleright$  Se repite a lo sumo la cantidad de char,
 $\text{que es una constate} \Rightarrow \mathcal{O}(1)$ 
    Avanzar(it)  $\triangleright \mathcal{O}(1)$ 
  res  $\leftarrow$  (HaySiguiente(it))  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$

Algorithm 56 Buscar

BUSCAR(in c : char, in/out t : diccT(string, σ) $\rightarrow res$: tupla<letra char, significado puntero(σ), prox diccT(string, σ)>

Pre $\equiv \{esta(c,t)\}$

Post $\equiv \{alias(res =_{obs} buscar(c,t))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Busca el elemento de la lista que este caracterizado con el char de entrada.

Aliasing: El resultado queda ligado con aliasing.

```

procedure BUSCAR(in  $c$ : char, in  $t$ : diccT(string, $\sigma$ )  $\rightarrow res$ : tupla < char, puntero( $\sigma$ ), diccT(string, $\sigma$ ) >
     $it \leftarrow crearIt(t)$ 
    while Siguiente(it).letra  $\neq c$  do           ▷ Se repite a lo sumo la cantidad de char, que es una constate  $\Rightarrow \mathcal{O}(1)$ 
        Avanzar(it)                               ▷  $\mathcal{O}(1)$ 
     $res \leftarrow Siguiente(it)$ 

```

Complejidad: $\mathcal{O}(1)$

Algorithm 57 iObtener

```

procedure IOBTENER(in/out  $e$ : estr, in  $c$ : string)  $\rightarrow res$ :  $\sigma$ 
     $piso \leftarrow e.trie$                                ▷  $\mathcal{O}(1)$ 
     $i \leftarrow 0$ 
    while  $i < |c| - 1$  do                               ▷ Se repite  $|c| - 1$  veces y lo de adentro es  $\mathcal{O}(1) \Rightarrow \mathcal{O}(|c|)$ 
         $piso \leftarrow Buscar(c[i],piso).prox$            ▷  $\mathcal{O}(1)$ 
         $i++$                                              ▷  $\mathcal{O}(1)$ 
     $res \leftarrow *(Buscar(c[i],piso).significado)$      ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(|c|)$

Algorithm 58 iBorrar

```

procedure IBORRAR(in/out  $e$ : estr, in  $c$ : string)
    Borrar(e.lineal, c)                               ▷  $\mathcal{O}(\#claves(e.lineal))$ 
     $piso \leftarrow e.trie$                                ▷  $\mathcal{O}(1)$ 
     $i \leftarrow 0$ 
    while  $i < |c| - 1$  do                               ▷ Se repite  $|c| - 1$  veces y lo de adentro es  $\mathcal{O}(1) \Rightarrow \mathcal{O}(|c|)$ 
         $piso \leftarrow Buscar(c[i],piso).prox$            ▷  $\mathcal{O}(1)$ 
         $i++$                                              ▷  $\mathcal{O}(1)$ 
    delete(Buscar(s[i],piso).significado)
    Buscar(s[i],piso).significado  $\leftarrow NULL$        ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(|c| + \#claves(e.lineal))$

Algorithm 59 iBorrarRapido

```

procedure IBORRARRAPIDO(in/out  $e$ : estr, in  $it$ : itDiccNom(string, $\sigma$ ))
    EliminarSiguiente(it)                             ▷  $c$  es la clave asociada al siguiente del iterador  $\mathcal{O}(|c|)$ 

```

Complejidad: $\mathcal{O}(|clave|)$ donde clave es la clave del siguiente del iterador.

Algorithm 60 i#Claves

```

procedure I#CLAVES(in  $e$ : estr)
     $res \leftarrow \#claves(e.lineal)$                                ▷  $\mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$

2.3.1. Iterador de Diccionario de Nombres

Interfaz

parámetros formales

géneros σ

función $\text{COPIAR}(\text{in } s : \sigma) \rightarrow \text{res} : \sigma$

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

Post $\equiv \{\text{res} =_{\text{obs}} a\}$

Complejidad: $\Theta(\text{copy}(s))$

Descripción: función de copia de σ 's

se explica con: ITERADOR UNIDIRECCIONAL MODIFICABLE (TUPLA<STRING, σ >)

usa: nat, bool, DiccionarioNombre(string, σ)

géneros: itDiccNom(string, σ)

Operaciones de DiccionarioNombres(string, σ)

$\text{CREARIT}(\text{in } d : \text{DiccNom}(\text{string}, \sigma)) \rightarrow \text{res} : \text{itDiccNom}(\text{string}, \sigma)$

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

Post $\equiv \{\text{alias}(\text{esPermutacion}(\text{secuSuby}(\text{res}), d)) \wedge \text{vacía}?(Anteriores(\text{res}))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve un iterador al diccionario

Aliasing: El iterador se invalida si se elimina el siguiente sin utilizar la función EliminarSiguiente. Los anteriores y siguientes pueden modificarse sin que se invalide el iterador

$\text{HAYSIGUIENTE}(\text{in } it : \text{itDiccNom}(\text{string}, \sigma)) \rightarrow \text{res} : \text{bool}$

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

Post $\equiv \{\text{res} =_{\text{obs}} \text{hayMas}(it)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve verdadero si y sólo si el iterador tiene un siguiente elemento

$\text{SIGUIENTE}(\text{in } it : \text{itDiccNom}(\text{string}, \sigma)) \rightarrow \text{res} : \text{tupla}<\text{string}, \sigma>$

Pre $\equiv \{\text{hayMas}(it)\}$

Post $\equiv \{\text{alias}(\text{res} =_{\text{obs}} \text{prim}(\text{Siguiente}(it)))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve el siguiente del iterador

Aliasing: res.siguiente es modificable si y solo si el iterador es modificable. res.clave no es modificable

$\text{SIGUIENTECLAVE}(\text{in } it : \text{itDiccNom}(\text{string}, \sigma)) \rightarrow \text{res} : \text{string}$

Pre $\equiv \{\text{hayMas}(it)\}$

Post $\equiv \{\text{res} =_{\text{obs}} \Pi_1(\text{prim}(\text{Siguiente}(it)))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve la siguiente clave

Aliasing: res no es modificable

$\text{SIGUIENTESIGNIFICADO}(\text{in } it : \text{itDiccNom}(\text{string}, \sigma)) \rightarrow \text{res} : \sigma$

Pre $\equiv \{\text{hayMas}(it)\}$

Post $\equiv \{\text{alias}(\text{res} =_{\text{obs}} \Pi_2(\text{prim}(\text{Siguiente}(it))))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve el siguiente significado del iterador

Aliasing: res es modificable si y solo si el iterador es modificable

$\text{AVANZAR}(\text{in/out } it : \text{itDiccNom}(\text{string}, \sigma))$

Pre $\equiv \{\text{hayMas}(it) \wedge it = it_0\}$

Post $\equiv \{it = \text{avanzar}(it_0)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Avanza una posición el iterador

ELIMINARSIGUIENTE(**in/out** *it*: itDiccNom(string, σ))
Pre $\equiv \{\text{hayMas}(it) \wedge it = it_0\}$
Post $\equiv \{it =_{\text{obs}} \text{Eliminar}(it_0)\}$
Complejidad: $\mathcal{O}(|c|)$ donde c es la clave asociada al siguiente del iterador
Descripción: Elimina el siguiente del iterador

Representación

Representación del iterador

itDiccNom(string, σ) se representa con *ite*

Donde *ite* es *it* : itDicc(string, σ)

Rep : *ite* \rightarrow bool

Rep(*i*) $\equiv \text{true} \iff \text{Rep}(\text{ite.it})$

Abs : *ite i* \rightarrow itDiccNom(string, σ)

Abs(*i*) $\equiv \text{Abs}(i.\text{it})$

{Rep(*i*)}

Algorithm 61 iCrearIt

procedure ICLEARIT(**in** *e* : **estr**) \rightarrow *res* : *ite*
 res \leftarrow crearIt(*e*.lineal)

$\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 62 iHaySiguiente

procedure IHAYSIGUIENTE(**in** *it* : **ite**) \rightarrow *res* : bool
 res \leftarrow haySiguiente(*ite*)

$\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 63 iSiguiente

procedure ISIGUIENTE(**in** *it* : **ite**) \rightarrow *res* : *tupla* \langle string, σ \rangle
 res \leftarrow Siguiente(*ite*)

$\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 64 iSiguienteClave

procedure ISIGUIENTECLAVE(**in** *it* : **ite**) \rightarrow *res* : string
 res \leftarrow Π_1 (Siguiente(*ite*))

$\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 65 iSiguienteSignificado

procedure ISIGUIENTESIGNIFICADO(**in** *it* : **ite**) \rightarrow *res* : σ
 res \leftarrow Π_2 (Siguiente(*ite*))

$\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 66 iAvanzar

procedure IAVANZAR(**in/out** *it* : **ite**)
 Avanzar(*ite*)

$\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 67 iEliminarSiguiente

```

procedure iELIMINARSIGUIENTE(in it : ite)  $\rightarrow$  res : tupla < string,  $\sigma$  >
    c  $\leftarrow$  SiguienteClave(ite)  $\triangleright \mathcal{O}(1)$ 
    EliminarSiguiente(ite)  $\triangleright \mathcal{O}(1)$ 
    piso  $\leftarrow$  e.trie  $\triangleright \mathcal{O}(1)$ 
    i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
    while i < |c|-1 do  $\triangleright$  Se repite |c| - 1 veces y lo de adentro es  $\mathcal{O}(1) \Rightarrow \mathcal{O}(|c|)$ 
        piso  $\leftarrow$  Buscar(c[i],piso).prox  $\triangleright \mathcal{O}(1)$ 
        i++  $\triangleright \mathcal{O}(1)$ 
    delete(Buscar(s[i],piso).significado)
    Buscar(s[i],piso).significado  $\leftarrow$  NULL  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$

2.3.2. Iterador de Claves de Diccionario de Nombres

Interfaz

se explica con: ITERADOR UNIDIRECCIONAL (STRING)

usa: nat, bool, DiccionarioNombre(string, σ), IteradorDiccionarioNombres(tupla<string, σ >)

géneros: itClavesDiccN(string)

Operaciones de Iterador de Claves de DiccionarioNombres(string)

CREARIT(in/out d : diccNom(string, σ)) $\rightarrow res$: itClavesDiccN(string)

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

Post $\equiv \{res =_{\text{obs}} \text{CrearItUni}(\text{Secuenciar}(\text{claves}(d)))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Crea un iterador no modificable de las claves del diccionario de entrada

Secuenciar : conj(string) \rightarrow secu(string)

Secuenciar(cs) \equiv if $\emptyset?(cs)$ then $<>$ else dameUno(cs) • Secuenciar(sinUno(cs)) fi

HAYSIGUIENTE(in it : itClavesDiccN(tupla<string, σ >)) $\rightarrow res$: bool

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

Post $\equiv \{res =_{\text{obs}} \text{HayMas?}(it)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve verdadero si y sólo si el iterador tiene un siguiente elemento

SIGUIENTECLAVE(in it : itClavesDiccN(tupla<string, σ >)) $\rightarrow res$: string

Pre $\equiv \{\text{HayMas?}(it)\}$

Post $\equiv \{res =_{\text{obs}} \Pi_1(\text{Actual}(it))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve la siguiente clave del diccionario

Aliasing: res no es modificable

AVANZAR(in/out it : itClavesDiccN(tupla<string, σ >))

Pre $\equiv \{\text{HayMas?}(it) \wedge it = it_0\}$

Post $\equiv \{it =_{\text{obs}} \text{Avanzar}(it_0)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Avanza el iterador

Representación

itClavesDiccN(string) se representa con ite

Donde ite es it : itDiccNom(tupla<string, σ >)

Rep : ite \rightarrow bool

Rep(i) $\equiv \text{true} \iff \text{Rep}(ite.it)$

Abs : ite $i \rightarrow$ itClavesDiccN(string)

{Rep(i)}

$$\text{Abs}(i) \equiv \text{Abs}(i.it)$$

Algoritmos

Algorithm 68 iCrearIt

```
procedure ICLEARIT(in d : diccNom(string,σ)) → res : ite  
    res ← CrearIt(d) // crearIt del iterador de Diccionario Nombre
```

▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 69 iHaySiguiente

```
procedure IHAYSIGUIENTE(in it : ite) → res : bool  
    res ← HaySiguiente(it)
```

▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 70 iSiguienteClave

```
procedure ISIGUIENTECLAVE(in it : ite) → res : string  
    res ← SiguienteClave(it)
```

▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 71 iAvanzar

```
procedure IAVANZAR(in/out it : ite)  
    Avanzar(it)
```

▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

2.4. Diccionario Logaritmico

Interfaz

parámetros formales

géneros σ
función $\text{COPIAR}(\text{in } s : \sigma) \rightarrow \text{res} : \sigma$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{\text{res} =_{\text{obs}} a\}$
Complejidad: $\Theta(\text{copy}(s))$
Descripción: función de copia de σ 's

se explica con: $\text{DICCIONARIOACOTADO}(\text{NAT}, \sigma)$

usa: nat, bool

géneros: $\text{diccLog}(\text{nat}, \sigma)$

Operaciones de $\text{DiccionarioLog}(\text{nat}, \sigma)$

$\text{VACIO}(\text{in } \text{tam} : \text{nat}) \rightarrow \text{res} : \text{diccLog}(\text{nat}, \sigma)$

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

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

Complejidad: $\mathcal{O}(1)$

Descripción: Crea un diccionario vacío con tam como límite de claves.

$\text{DEFINIR}(\text{in/out } d : \text{diccLog}(\text{nat}, \sigma), \text{in } c : \text{nat}, \text{in } s : \sigma) \rightarrow \text{res} : \text{itDiccLog}(\text{tupla}\langle \text{nat}, \sigma \rangle)$

Pre $\equiv \{(\forall c' : \text{nat})(c' \in \text{claves}(d) \Rightarrow c > c') \wedge d = d_0 \wedge \#\text{claves}(d) < \text{tamaño}(d)\}$

Post $\equiv \{d =_{\text{obs}} \text{Definir}(c, s, d_0) \wedge \text{alias}(\text{Siguiete}(\text{res}) =_{\text{obs}} \text{tupla}\langle c, s \rangle)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Agrega la clave c con significado s, y devuelve un iterador apuntando a esta nueva definición.

Aliasing: Del siguiente de res sólo puedo modificarse el significado (si σ es modificable).

$\text{DEFINIRLENTO}(\text{in/out } d : \text{diccLog}(\text{nat}, \sigma), \text{in } c : \text{nat}, \text{in } s : \sigma) \rightarrow \text{res} : \text{itDiccLog}(\text{tupla}\langle \text{nat}, \sigma \rangle)$

Pre $\equiv \{\neg \text{def}(c, d) \wedge d = d_0 \wedge \#\text{claves}(d) \leq \text{tamaño}(d)\}$

Post $\equiv \{d =_{\text{obs}} \text{definir}(c, s, d_0) \wedge \text{alias}(\text{Siguiete}(\text{res}) =_{\text{obs}} \text{tupla}\langle c, s \rangle)\}$

Complejidad: $\mathcal{O}(\#\text{claves}(d))$

Descripción: Agrega la clave c con significado s, y devuelve un iterador apuntando a esta nueva definición.

Aliasing: Del siguiente de res sólo puedo modificarse el significado (si σ es modificable).

$\text{DEFINIDO}(\text{in/out } d : \text{diccLog}(\text{nat}, \sigma), \text{in } c : \text{nat}) \rightarrow \text{res} : \text{bool}$

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

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

Complejidad: $\mathcal{O}(\log \#\text{claves}(d))$

$\text{OBTENER}(\text{in/out } d : \text{diccLog}(\text{nat}, \sigma), \text{in } c : \text{nat}) \rightarrow \text{res} : \sigma$

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

Post $\equiv \{\text{alias}(\text{res} =_{\text{obs}} \text{obtener}(c, d)) \wedge (\forall c' : \text{nat})(c' \in \text{claves}(d) \wedge c \neq c') \Rightarrow_{\text{L}} \text{obtener}(c', d) =_{\text{obs}} \text{obtener}(c, d_0)\}$

Complejidad: $\mathcal{O}(\log \#\text{claves}(d))$

Descripción: Devuelve el significado de la clave c.

Aliasing: res es modificable si y sólo si σ modificable.

Representación

Representación DiccionarioLog(nat, σ)

diccLog(nat, σ) se representa con estr

donde **estr** es tupla(*tamaño*: nat
valores: arreglo dimensionable de tupla $\langle \text{clave} : \text{nat}, \text{significado} : \sigma \rangle$, *proxA-*
Definir: nat)

Rep : estr \rightarrow bool

Rep(*e*) \equiv true \iff SinRepetidos(Claves(*e.valores*)) \wedge Ordenada(Claves(*e.valores*))
 \wedge *e.tamaño* =_{obs} tamaño(*e.valores*) \wedge *e.proxADefinir* \leq *e.tamaño*
 \wedge ($\forall i : \text{nat}$) ($i < \text{e.tamaño} \wedge \neg \text{definido}(\text{e.valores}, i) \Rightarrow_{\text{L}}$
 $(\forall j : \text{nat}) (i < j \wedge j < \text{e.tamaño}) \Rightarrow_{\text{L}} \neg \text{definido}(\text{e.valores}, j))$)

SinRepetidos : secu(nat) \rightarrow bool

SinRepetidos(*s*) \equiv **if** vacío?(*s*) **then** true **else** \neg Esta?(prim(*s*),fin(*s*)) \wedge sinRepetidos(fin(*s*)) **fi**

val es arreglo dimensionable de tupla $\langle \text{clave} : \text{nat}, \text{significado} : \sigma \rangle$

Claves : val \rightarrow secu(nat)

Claves(*v*) \equiv **if** tamaño(*v*) = 0 **then** $\langle \rangle$ **else** Primero(*v*).clave • Claves(fin(*v*)) **fi**

Ordenada : secu(nat) \rightarrow bool

Ordenada(*s*) \equiv **if** vacía?(*s*) **then** true **else** ($\forall n : \text{nat}$) ($n \in \text{fin}(\text{s}) \Rightarrow n > \text{Primero}(\text{s}) \wedge \text{Ordenada}(\text{fin}(\text{s}))$) **fi**

Abs : estr *e* \rightarrow diccLog(nat, σ)

{Rep(*e*)}

Abs(*e*) \equiv *d* : diccLog(nat, σ)/

e.tamaño =_{obs} tamaño(*d*)

($\forall c : \text{nat}$) ($\text{def?}(c,d) \iff (\exists i : \text{nat}) (0 \leq i < \text{e.tamaño} \wedge \text{definido}(\text{e.valores}, i) \wedge_{\text{L}} \text{e.valores}[i].\text{clave} =_{\text{obs}} c)$)
 $\wedge_{\text{L}} (\text{def}(c,d) \Rightarrow_{\text{L}} \text{Obtener}(c,d) =_{\text{obs}} \text{Significado}(c,e))$)

Significado : nat \times estr \rightarrow σ

{def?(*c*,*d*)}

Significado(*c*,*e*) \equiv *e.valores*[Posicion(*c*,*e.valores*)].significado

Posicion : nat \times val \rightarrow nat

Posicion(*c*,*v*) \equiv **if** Primero(*v*).clave = *c* **then** 0 **else** 1 + Posicion(*c*,fin(*v*)) **fi**

Algorithm 72 iVacio

procedure IVACIO(in tam : nat) \rightarrow res : estr

e.tam \leftarrow tam

$\triangleright \mathcal{O}(1)$

e.valores \leftarrow CrearArreglo(tam)

$\triangleright \mathcal{O}(\text{tam})$

e.proxADefinir \leftarrow 0

$\triangleright \mathcal{O}(\text{tam})$

Complejidad: $\mathcal{O}(1)$

Algorithm 73 iDefinir

procedure IDEFINIR(in/out *e* : estr, in *c* : nat, in *s* : σ) \rightarrow res : ite

e.valores[*e.proxADef*].clave \leftarrow *c*

$\triangleright \mathcal{O}(1)$

e.valores[*e.proxADef*].significado \leftarrow Copiar(*s*)

$\triangleright \mathcal{O}(\text{copy}(\text{s}))$

res.pos \leftarrow *e.proxADefinir*

$\triangleright \mathcal{O}(1)$

res.dic \leftarrow puntero(*e*)

$\triangleright \mathcal{O}(1)$

e.proxADefinir \leftarrow *e.proxADefinir* + 1

$\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(\text{copy}(\text{s}))$

Algorithm 74 iDefinido

```

procedure IDEFINIDO(in e : estr, in c : nat)  $\rightarrow res : bool$ 
  arriba  $\leftarrow e.proxADefinir - 1$   $\triangleright \mathcal{O}(1)$ 
  abajo  $\leftarrow 0$   $\triangleright \mathcal{O}(1)$ 
  while arriba > abajo do  $\triangleright$  Se repite  $\log e.proxADefinir$  veces y lo de adentro es  $\mathcal{O}(1)$ 
     $\Rightarrow \mathcal{O}(\log e.proxADefinir)$ 
    medio  $\leftarrow (arriba + abajo) / 2$   $\triangleright \mathcal{O}(1)$ 
    if e.valores[medio].clave < c then  $\triangleright \mathcal{O}(1)$ 
      abajo  $\leftarrow medio$   $\triangleright \mathcal{O}(1)$ 
    else
      arriba  $\leftarrow medio$   $\triangleright \mathcal{O}(1)$ 
  res  $\leftarrow (e.valores[medio].clave = c)$   $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(\log \#claves)$ Justificación: $e.proxADefinir - 1$ coincide con la cantidad de claves definidas en el diccionario, pues se van agregando adelante. $\mathcal{O}(e.proxADefinir) = \mathcal{O}(\#claves)$ **Algorithm 75** iDefinirLento

```

procedure IDEFINIRLENTO(in/out e : estr, in c : nat, in s :  $\sigma$ )  $\rightarrow res : ite$ 
  i  $\leftarrow e.proxADefinir$   $\triangleright \mathcal{O}(1)$ 
  while i > 0  $\wedge e.valores[i-1].clave > c$  do  $\triangleright$  En el peor caso se repite  $e.proxADefinir$  veces,
     $y lo de adentro es \mathcal{O}(\text{copy}(s)) \Rightarrow \mathcal{O}(\text{copy}(s) * e.proxADefinir)$ 
    e.valores[i].clave  $\leftarrow e.valores[i-1].clave$   $\triangleright \mathcal{O}(1)$ 
    e.valores[i].significado  $\leftarrow e.valores[i-1].significado$   $\triangleright \mathcal{O}(\text{copy}(s))$ 
    i  $\leftarrow i - 1$   $\triangleright \mathcal{O}(1)$ 
  e.valores[i].clave  $\leftarrow c$   $\triangleright \mathcal{O}(1)$ 
  e.valores[i].significado  $\leftarrow \text{Copiar}(s)$   $\triangleright \mathcal{O}(\text{copy}(s))$ 
  res.pos  $\leftarrow e.proxADefinir$   $\triangleright \mathcal{O}(1)$ 
  res.dic  $\leftarrow \text{puntero}(e)$   $\triangleright \mathcal{O}(1)$ 
  e.proxADefinir  $\leftarrow e.proxADefinir + 1$   $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(\#claves * \text{copy}(s))$ Justificación: $e.proxADefinir - 1$ coincide con la cantidad de claves definidas en el diccionario, pues se van agregando adelante. $\mathcal{O}(e.proxADefinir) = \mathcal{O}(\#claves)$. $\mathcal{O}(1) + \mathcal{O}(\text{copy}(s)) + \mathcal{O}(\text{copy}(s) * e.proxADefinir) = \mathcal{O}(\text{copy}(s) * e.proxADefinir) = \mathcal{O}(\#claves * \text{copy}(s))$ **Algorithm 76** iObtener

```

procedure IOBTENER(in e : estr, in c : nat)  $\rightarrow res : \sigma$ 
  arriba  $\leftarrow e.proxADefinir - 1$   $\triangleright \mathcal{O}(1)$ 
  abajo  $\leftarrow 0$   $\triangleright \mathcal{O}(1)$ 
  while arriba > abajo do  $\triangleright$  Se repite  $\log e.proxADefinir$  veces y lo de adentro es  $\mathcal{O}(1)$ 
     $\Rightarrow \mathcal{O}(\log e.proxADefinir)$ 
    medio  $\leftarrow (arriba + abajo) / 2$   $\triangleright \mathcal{O}(1)$ 
    if e.valores[medio].clave < c then  $\triangleright \mathcal{O}(1)$ 
      abajo  $\leftarrow medio$   $\triangleright \mathcal{O}(1)$ 
    else
      arriba  $\leftarrow medio$   $\triangleright \mathcal{O}(1)$ 
  res  $\leftarrow e.valores[medio].significado$   $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(\log \#claves)$ Justificación: $e.proxADefinir - 1$ coincide con la cantidad de claves definidas en el diccionario, pues se van agregando adelante. $\mathcal{O}(e.proxADefinir) = \mathcal{O}(\#claves)$

2.4.1. Iterador de Diccionario Logaritmico

Interfaz

parámetros formales

géneros σ

función $\text{COPIAR}(\text{in } s : \sigma) \rightarrow \text{res} : \sigma$

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

Post $\equiv \{\text{res} =_{\text{obs}} a\}$

Complejidad: $\Theta(\text{copy}(s))$

Descripción: función de copia de σ 's

se explica con: ITERADOR UNIDIRECCIONAL MODIFICABLE(TUPLA<NAT, σ >)

usa: nat, bool, diccionarioLog(nat, σ)

géneros: itDiccLog(tupla<nat, σ >)

Operaciones de Iterador DiccionarioLog(nat, σ)

HAYSIGUIENTE(in it: itDiccLog(tupla<nat, σ >)) $\rightarrow \text{res} : \text{bool}$

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

Post $\equiv \{\text{res} =_{\text{obs}} \text{HayMas?}(it)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve verdadero si y sólo si quedan elementos que recorrer en el iterador.

SIGUIENTECLAVE(in it: itDiccLog(tupla<nat, σ >)) $\rightarrow \text{res} : \text{nat}$

Pre $\equiv \{\text{HayMas?}(it)\}$

Post $\equiv \{\text{res} =_{\text{obs}} \Pi_1(\text{Siguiente}(it))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve la clave del siguiente elemento del iterador.

Aliasing: res no es modificable.

SIGUIENTESIGNIFICADO(in/out it: itDiccLog(tupla<nat, σ >)) $\rightarrow \text{res} : \sigma$

Pre $\equiv \{\text{HayMas?}(it)\}$

Post $\equiv \{\text{alias}(\text{res} =_{\text{obs}} \Pi_2(\text{Siguiente}(it)))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve el siguiente del siguiente elemento del iterador.

Aliasing: res es modificable si y sólo si σ es modificable.

Representación

itDiccLog(tupla <nat, σ >) se representa con ite

donde ite es tupla(pos: nat, dic: puntero(estr))

donde estr es la representación del diccionarioLog(nat, σ)

Rep : ite $\rightarrow \text{bool}$

Rep(i) $\equiv \text{true} \iff i.\text{pos} \leq *(i.\text{dic}).\text{proxADefinir}$

Abs : ite i $\rightarrow \text{itDiccLog}(\text{tupla}<\text{nat}, \sigma>)$

{Rep(i)}

Abs(i) $=_{\text{obs}} \text{itL: itDiccLog}(\text{tupla}<\text{nat}, \sigma>) \mid \text{Siguientes}(\text{itL}) =_{\text{obs}}$

SecuenciarClavesArreglo(*(i.dic).valores, *(i.dic).tamaño, i.pos)

SecuenciarClavesArreglo : arreglo(tupla<clave:nat \times significado: σ >) $\times \text{nat} \rightarrow \text{secu}(\text{nat})$

```

SecuenciaClavesArreglo(var, tam, pos)  $\equiv$  if pos =obs tam then
    <>
else
    var[pos].clave • SecuenciarArreglo(var, tam, pos + 1 )
fi

```

Algoritmos

Algorithm 77 iHaySiguiente

```

procedure IHAYSIGUIENTE(in i : ite)  $\rightarrow$  res : bool
    res  $\leftarrow$  (i.pos < *(i.dic).tamaño)

```

▷ $\mathcal{O}(1)$ Complejidad: $\mathcal{O}(1)$

Algorithm 78 iSiguienteClave

```

procedure ISIGUIENTECLAVE(in i : ite)  $\rightarrow$  res : nat
    res  $\leftarrow$  *(i.dic).valores[i.pos].clave

```

▷ $\mathcal{O}(1)$ Complejidad: $\mathcal{O}(1)$

Algorithm 79 iSiguienteSignificado

```

procedure ISIGUIENTESIGNIFICADO(in i : ite)  $\rightarrow$  res : nat
    res  $\leftarrow$  *(i.dic).valores[i.pos].significado

```

▷ $\mathcal{O}(1)$ Complejidad: $\mathcal{O}(1)$

2.4.2. Iterador de Claves de Diccionario Logaritmico

Interfaz

se explica con: ITERADOR UNIDIRECCIONAL (NAT)

usa: nat, bool, DiccionarioLog(nat, σ)

géneros: itClavesDiccLog(nat)

Operaciones de Iterador de Claves de DiccionarioPlacas(nat)

CREARIT(**in** d: diccLog(nat, σ)) \rightarrow res : itClavesDiccLog(nat)**Pre** \equiv {true}**Post** \equiv {Permutacion(Siguientes(res), Secuenciar(claves(d)))}**Complejidad:** $\mathcal{O}(1)$ **Descripción:** Crea un iterador no modificable de las claves del diccionario de entrada. El iterador se invalida si se modifica el diccionario.Secuenciar : conj(string) \rightarrow secu(string)Secuenciar(cs) \equiv **if** $\emptyset?$ (cs) **then** <> **else** dameUno(cs) • Secuenciar(sinUno(cs)) **fi**Permutacion : secu(nat) \times secu(nat) \rightarrow boolPermutacion(s,t) \equiv Iguales(MultiConjuntizar(s), MultiConjuntizar(t))MultiConjuntizar : secu(nat) \rightarrow multiconj(nat)MultiConjuntizar(s) \equiv **if** vacio?(s) **then** \emptyset **else** Ag(prim(s), MultiConjuntizar(fin(s))) **fi**

Iguales : multiconj(nat) \times multiconj(nat) \rightarrow bool

Iguales(m_1, m_2) \equiv **if** $\#m_1 = 0$ **then**
 $(\#m_2 = 0)$
 else
 $\#(\text{dameUno}(m_1), m_1) = \#(\text{dameUno}(m_1), m_2)$
 $\wedge \text{Igual}(\text{sinUno}(m_1), m_2 - \{\text{dameUno}(m_1)\})$
 fi

HAYSIGUIENTE(**in** it : itClavesDiccLog(nat)) $\rightarrow res$: bool

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

Post $\equiv \{res =_{\text{obs}} \text{HayMas?}(it)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve verdadero si y sólo si el iterador tiene un siguiente elemento

SIGUIENTECLAVE(**in** it : itClavesDiccLog(nat)) $\rightarrow res$: string

Pre $\equiv \{\text{HayMas?}(it)\}$

Post $\equiv \{res =_{\text{obs}} \Pi_1(\text{Actual}(it))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve la siguiente clave del diccionario.

Aliasing: res no es modificable.

AVANZAR(**in/out** it : itClavesDiccLog(nat))

Pre $\equiv \{\text{HayMas?}(it) \wedge it = it_0\}$

Post $\equiv \{it =_{\text{obs}} \text{Avanzar}(it_0)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Avanza el iterador.

Representación

itClavesDiccLog(nat) se representa con ite

donde ite es $\text{tupla}(pos: \text{nat}, dic: \text{puntero}(\text{estr}))$

donde $estr$ es la representación del diccionarioLog

Rep : ite \rightarrow bool

Rep(i) $\equiv \text{true} \iff i.pos \leq *(i.dic).proxADefinir \wedge i.dic \neq \text{NULL}$

Abs : ite $i \rightarrow$ itClavesDiccLog(nat)

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

Abs(i) =_{obs} itC: itClavesDiccLog(nat) |

Siguientes(itC) =_{obs} SecuenciarClavesArreglo($*(i.dic).valores,$
 $*(i.dic).tamaño, i.pos$)

SecuenciarClavesArreglo : arreglo($\text{tupla}(\text{clave: nat} \times \text{significado: } \sigma) \times \text{nat} \rightarrow \text{secu}(\text{nat})$

SecuenciaClavesArreglo(var, tam, pos) \equiv **if** $pos =_{\text{obs}} \text{tam}$ **then**

$\langle \rangle$

else

var[pos].clave • SecuenciarArreglo(var, tam, pos + 1)

fi

Algoritmos

Algorithm 80 iCrearIt

procedure ICREARIT(**in** e : estr) $\rightarrow res$: ite

$res.pos \leftarrow 0$

$\triangleright \mathcal{O}(1)$

$res.dic \leftarrow \text{puntero}(e)$

$\triangleright \mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Algorithm 81 iHaySiguiente

```
procedure IHAYSIGUIENTE(in i : ite)  $\rightarrow$  res : bool  
    res  $\leftarrow$  ( i.pos < *(i.dic).tamaño )  $\triangleright \mathcal{O}(1)$ 
```

Complejidad: $\mathcal{O}(1)$

Algorithm 82 iSiguienteClave

```
procedure ISIGUIENTECLAVE(in i : ite)  $\rightarrow$  res : nat  
    res  $\leftarrow$  *(i.dic).valores[i.pos]  $\triangleright \mathcal{O}(1)$ 
```

Complejidad: $\mathcal{O}(1)$

Algorithm 83 iAvanzar

```
procedure IAVANZAR(in/out i : ite)  
    i.pos  $\leftarrow$  i.pos + 1  $\triangleright \mathcal{O}(1)$ 
```

Complejidad: $\mathcal{O}(1)$

2.5. Diccionario_H

Interfaz

parámetros formales

géneros σ

usa: nat, bool, arreglo Dimensionable

géneros: dicch(nat, σ)

se explica con: DICCIONARIOACOTADO(NAT, σ)

Operaciones de Dicch(nat, σ)

VACIO(**in** cant : nat) \rightarrow res : dicch(nat, σ)

Pre $\equiv \{ \text{cant} > 0 \}$

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

Complejidad: $\mathcal{O}(\text{cant})$

Descripción: Crea un diccionario vacio de cant posiciones.

DEFINIR(**in** clave : nat, **in** significado : σ , **in/out** dicc : dicch(nat, σ))

Pre $\equiv \{ \text{dicc} = \text{dicc}_0 \wedge \neg \text{def?}(\text{clave}, \text{dicc}_0) \wedge \text{claves}(\text{dicc}_0) < \text{tamaño}(\text{dicc}_0) \}$

Post $\equiv \{ \text{dicc} =_{\text{obs}} \text{Definir}(\text{clave}, \text{significado}, \text{dicc}_0) \}$

Complejidad: $\mathcal{O}(1)$ en caso promedio

Descripción: Agrega un elemento al diccionario.

OBTENER(**in** clave : nat, **in/out** dicc : dicch(nat, σ)) \rightarrow res : σ

Pre $\equiv \{ \text{def?}(\text{clave}, \text{dicc}_0) \wedge \text{dicc} = \text{dicc}_0 \}$

Post $\equiv \{ \text{alias}(\text{res} =_{\text{obs}} \text{obtener}(\text{clave}, \text{dicc})) \wedge$

$(\forall \text{clave}' : \text{nat}) ((\text{clave}' \in \text{claves}(\text{dicc}) \wedge \text{clave} \neq \text{clave}') \Rightarrow_{\text{L}} \text{obtener}(\text{clave}', \text{dicc}) =_{\text{obs}} \text{obtener}(\text{clave}', \text{dicc}_0)) \}$

Complejidad: $\mathcal{O}(1)$ en caso promedio

Representación

Representacion del DiccionarioHash(nat, σ)

dicch(nat, σ) se representa con estr

donde estr es tupla(cant : nat, valores : Adt)

Donde Adt es arreglo Dimensionable de tupla<clave : nat, significado : σ >

Rep : estr \rightarrow bool

Rep(e) $\equiv \text{true} \iff \text{e.cant} =_{\text{obs}} \text{Tamaño}(\text{e.valores}) \wedge_{\text{L}}$
 $(\forall i, j : \text{nat}) (i < \text{e.cant} \wedge j < \text{e.cant} \wedge i \neq j \wedge \text{definido}(\text{e.valores}, i) \wedge \text{definido}(\text{e.valores}, j))$
 $\Rightarrow_{\text{L}} \text{e.valores}[i].\text{clave} \neq \text{e.valores}[j].\text{clave}$

Abs : estr e \rightarrow dicch(nat, σ)

{Rep(e)}

Abs(e) $\equiv \text{dic} : \text{dicch}(\text{nat}, \sigma) /$

$(\forall \text{clave} : \text{nat})$

$(\text{Def?}(\text{clave}, \text{dic}) \iff (\exists i : \text{nat}) (0 \leq i < \text{e.cant} \wedge \text{Definido}(\text{e.valores}, i) \wedge_{\text{L}} \text{e.valores}[i].\text{clave} = \text{clave}))$

$\wedge (\text{Def?}(\text{clave}, \text{dic})$

$\Rightarrow_{\text{L}} \text{Obtener}(\text{clave}, \text{dic}) =_{\text{obs}} \text{e.valores}[\text{mod}(\text{hash}(\text{clave}, \text{e}) + \text{corrimiento}(\text{hash}(\text{clave}, \text{e}), \text{e}), \text{cant})].\text{signi-}$

ficado

hash : nat \times estr \rightarrow nat

hash(clave, e) $\equiv \text{mod}(\text{clave}, \text{e.cant})$

corrimiento : nat \times estr \rightarrow nat

```

corrimiento(clave, e)  $\equiv$  if e.valores[ mod( clave, e.cant) ].clave = clave then
    0
else
    1 + corrimiento(clave + 1, e)
fi

```

Algoritmos

Algorithm 84 iVacio

```

procedure iVACIO(in n : nat)  $\rightarrow$  res : estr
    var e : estr
    e.cant  $\leftarrow$  n  $\triangleright \mathcal{O}(1)$ 
    e.valores  $\leftarrow$  CrearArreglo(n)  $\triangleright \mathcal{O}(n)$ 
    res  $\leftarrow$  e  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(n)$

Algorithm 85 iDefinir

```

procedure iDEFINIR(in clave : nat, in significado :  $\sigma$ , in/out e : estr)
    var h : nat
    h  $\leftarrow$  mod(clave, e.cant)  $\triangleright \mathcal{O}(1)$ 
    while Definido(e.valores, h) do  $\triangleright$  En promedio no se entra al ciclo
        h  $\leftarrow$  mod(h+1, e.cant)  $\triangleright \mathcal{O}(1)$ 
    e.valores[h]  $\leftarrow$  tupla<clave, significado>  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ promedio

Justificación: sabiendo que hay una distribución uniforme de 'claves', podemos asumir que en promedio no se entraría al ciclo, cerrando en promedio una suma de operaciones de complejidad $\mathcal{O}(1)$.

Algorithm 86 iObtener

```

procedure iOBTENER(in clave : nat, in/out e : estr)  $\rightarrow$  res :  $\sigma$ 
    var h : nat
    h  $\leftarrow$  mod(clave, e.cant)  $\triangleright \mathcal{O}(1)$ 
    while e.valores[h].clave  $\neq$  clave do  $\triangleright$  En promedio no se entra al ciclo
        h  $\leftarrow$  mod(h+1, e.cant)  $\triangleright \mathcal{O}(1)$ 
    res  $\leftarrow$  e.valores[h].significado  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$ promedio

Justificación: Misma justificación que el algoritmo anterior

2.6. Posicion

Interfaz

se explica con: TAD POSICION.

géneros: posicion.

Operaciones básicas de posicion

CREARPOSICION(in $x : \text{nat}$, in $y : \text{nat}$) $\rightarrow res : \text{posicion}$

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

Post $\equiv \{res =_{\text{obs}} \langle x, y \rangle\}$

Complejidad: $\Theta(1)$

Descripción: Crea una nueva instancia de posición haciendo copia por valor de los parámetros de entrada.

X(in $p : \text{posicion}$) $\rightarrow res : \text{nat}$

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

Post $\equiv \{\text{alias}(res =_{\text{obs}} X(p))\}$

Complejidad: $\Theta(1)$

Descripción: Devuelve el valor de la variable "X" de la posición

Aliasing: Hay aliasing entre el resultado de la función y el valor "X" de la instancia posicion.

Y(in $p : \text{posicion}$) $\rightarrow res : \text{nat}$

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

Post $\equiv \{\text{alias}(res =_{\text{obs}} Y(p))\}$

Complejidad: $\Theta(1)$

Descripción: Devuelve el valor de la variable "Y" de la posicion

Aliasing: Hay aliasing entre el resultado de la función y el valor "Y" de la instancia posicion.

SUMARPOSICIONES(in $p_1 : \text{posicion}$, in $p_2 : \text{posicion}$) $\rightarrow res : \text{posicion}$

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

Post $\equiv \{res =_{\text{obs}} \langle X(p_1) + X(p_2), Y(p_1) + Y(p_2) \rangle\}$

Complejidad: $\Theta(1)$

Descripción: Devuelve una nueva instancia del tipo posicion, resultado de sumar ambas X's e Y's.

RESTARPOSICIONES(in $p_1 : \text{posicion}$, in $p_2 : \text{posicion}$) $\rightarrow res : \text{posicion}$

Pre $\equiv \{Y(p_2) < Y(p_1) \wedge X(p_2) < X(p_1)\}$

Post $\equiv \{res =_{\text{obs}} \langle X(p_1) - X(p_2), Y(p_1) - Y(p_2) \rangle\}$

Complejidad: $\Theta(1)$

Descripción: Devuelve una nueva instancia del tipo posicion, resultado de restar a la primer posicion, la segunda.

DISTANCIAPOSICIONES(in $p_1 : \text{posicion}$, in $p_2 : \text{posicion}$) $\rightarrow res : \text{nat}$

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

Post $\equiv \{res =_{\text{obs}} \text{Max}(X(p_1), X(p_2)) - \text{Min}(X(p_1), X(p_2)) + \text{Max}(Y(p_1), Y(p_2)) - \text{Max}(Y(p_1), Y(p_2))\}$

Complejidad: $\Theta(1)$

Descripción: Devuelve el resultado de calcular la distancia entre dos instancias de posicion, en base a sus observadores X e Y.

Representación

Representación de posicion

posicion se representa con estr

donde estr es $\text{tupla}(x : \text{nat}, y : \text{nat})$

$\text{Rep} : \text{estr} \rightarrow \text{bool}$

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

$\text{Abs} : \text{estr } e \rightarrow \text{posicion}$

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

$\text{Abs}(e) =_{\text{obs}} p: \text{posicion} \mid X(p) = e.x \wedge Y(p) = e.y$

Algoritmos

Algorithm 87 iCrearPosicion

```

procedure ICREARPOSICION(in x : nat, in y : nat)  $\rightarrow$  res : estr
  var pos : estr
  pos.x  $\leftarrow$  x  $\triangleright \Theta(1)$ 
  pos.y  $\leftarrow$  y  $\triangleright \Theta(1)$ 
  res  $\leftarrow$  pos  $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(1)$

Algorithm 88 iX

```

procedure IX(in pos : estr)  $\rightarrow$  res : nat
  res  $\leftarrow$  pos.x  $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(1)$

Algorithm 89 iY

```

procedure IY(in pos : estr)  $\rightarrow$  res : nat
  res  $\leftarrow$  pos.y  $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(1)$

Algorithm 90 iSumarPosiciones

```

procedure ISUMARPOSICION(in pos1 : estr, in pos2 : estr)  $\rightarrow$  res : estr
  var pos : estr
  pos  $\leftarrow$  iCrearPosicion(pos1.x + pos2.x, pos1.y + pos2.y)  $\triangleright \Theta(1)$ 
  res  $\leftarrow$  pos  $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(1)$

Algorithm 91 iRestarPosiciones

```

procedure IRESTARPOSICIONES(in pos1 : estr, in pos2 : estr)  $\rightarrow$  res : estr
  var pos : estr
  pos  $\leftarrow$  iCrearPosicion(pos1.x - pos2.x, pos1.y - pos2.y)  $\triangleright \Theta(1)$ 
  res  $\leftarrow$  pos  $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(1)$

Algorithm 92 iDistanciaPosiciones

```

procedure IDISTANCIAPOSICIONES(in pos1 : estr, in pos2 : estr)  $\rightarrow$  res : nat
  var pos : nat
  pos  $\leftarrow$  Max(pos1.x, pos2.x) - Min(pos1.x, pos2.x) + Max(pos1.y, pos2.y) - Min(pos1.y, pos2.y)  $\triangleright \Theta(1)$ 
  res  $\leftarrow$  pos  $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(1)$

2.7. Matriz(α)

Interfaz

parámetros formales

géneros α
función COPIAR(**in** $a : \alpha \rightarrow res : \alpha$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} a\}$
Complejidad: $\Theta(\text{copy}(a))$
Descripción: función de copia de α 's

se explica con: MATRIZ(α).

géneros: matriz(α).

Operaciones básicas de matriz

CREARMATRIZ(**in** $alto : \text{nat}$, **in** $ancho : \text{nat}$, **in** $a : \alpha \rightarrow res : \text{matriz}(\alpha)$

Pre $\equiv \{ancho > 0 \wedge alto > 0\}$

Post $\equiv \{res =_{\text{obs}} \text{Crear}(ancho, alto, a)\}$

Complejidad: $\mathcal{O}(ancho \times alto \times \text{copy}(\sigma))$

Descripción: Genera una matriz de $ancho \times alto$ que contenga todos a .

DEFINIR(**in/out** $m : \text{matriz}(\alpha)$, **in** $ancho : \text{nat}$, **in** $alto : \text{nat}$, **in** $a : \alpha$)

Pre $\equiv \{\text{posValida}(m, ancho, alto) \wedge m_0 = m\}$

Post $\equiv \{m =_{\text{obs}} \text{definir}(alto, ancho, a, m_0)\}$

Complejidad: $\mathcal{O}(\text{copy}(\alpha))$

Descripción: Defino el elemento a en la matriz.

•[•][•](**in/out** $m : \text{matriz}(\alpha)$, **in** $ancho : \text{nat}$, **in** $alto : \text{nat}$) $\rightarrow res : \alpha$

Pre $\equiv \{\text{posValida}(m, ancho, alto)\}$

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{valor}(m, ancho, alto))\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve el elemento en la posición $ancho \times alto$ de la matriz m con alias.

Aliasing: res es modificable si y sólo si α es modificable.

COPIAR(**in** $m : \text{matriz}(\alpha) \rightarrow res : \text{matriz}(\alpha)$

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

Post $\equiv \{res =_{\text{obs}} m\}$

Complejidad: $\mathcal{O}(ancho \times alto \times \text{copy}(\alpha))$

Descripción: Crea una copia de la matriz m .

ALTO(**in** $m : \text{matriz}(\alpha) \rightarrow res : \text{nat}$

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

Post $\equiv \{res =_{\text{obs}} \text{alto}(m)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Retorna la altura de la matriz m .

ANCHO(**in** $m : \text{matriz}(\alpha) \rightarrow res : \text{nat}$

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

Post $\equiv \{res =_{\text{obs}} \text{ancho}(m)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Retorna el ancho de la matriz m .

POSVALIDA(**in** $m : \text{matriz}(\alpha)$, **in** $ancho : \text{nat}$, **in** $alto : \text{nat}$) $\rightarrow res : \text{nat}$

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

Post $\equiv \{res =_{\text{obs}} \text{posValida}(m, ancho, alto)\}$

Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve si es una posición válida de mi matriz o no.

Representación

Representación de la matriz

$\text{matriz}(\alpha)$ se representa con *estr*

donde *estr* es $\text{tupla}(\text{matriz: arreglo_dimensionable de arreglo_dimensionable de } \alpha$
 $, \text{ancho: nat}, \text{alto: nat})$

$\text{Rep} : \text{estr} \rightarrow \text{bool}$

$\text{Rep}(e) \equiv \text{true} \iff e.\text{ancho} > 0 \wedge e.\text{alto} > 0 \wedge \text{tamaño}(e.\text{matriz}) = e.\text{ancho} \wedge$
 $(\forall i : \text{nat})(i < e.\text{ancho} \Rightarrow_L \text{tamaño}(e.\text{matriz}[i]) = e.\text{alto})$

$\text{Abs} : \text{estr } e \rightarrow \text{matriz}(\alpha)$

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

$\text{Abs}(e) \equiv \text{mat: matriz}(\alpha) \mid e.\text{ancho} = \text{Ancho}(\text{mat}) \wedge e.\text{alto} = \text{Alto}(\text{mat}) \wedge_L$
 $(\forall i, j : \text{nat})(i < e.\text{ancho} \wedge j < e.\text{alto} \Rightarrow_L (e.\text{matriz}[i])[j] = \text{Valor?}(\text{mat}, i, j))$

Algoritmos

Algorithm 93 iCrearMatriz

```

1: procedure ICREAMATRIZ(in ancho : nat, in alto : nat, in a :  $\alpha$ )  $\rightarrow$  res : estr
2:   res  $\leftarrow$  CrearArreglo(ancho) de arreglos_dimensionables de  $\alpha$   $\triangleright \mathcal{O}(\text{ancho})$ 
3:   i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
4:   while i < ancho do  $\triangleright \mathcal{O}(\text{alto} \times \text{copy}(\alpha))$  ancho de veces  $\Rightarrow \mathcal{O}(\text{ancho} \times \text{alto} \times \text{copy}(\alpha))$ 
5:     j  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
6:     res[i]  $\leftarrow$  CrearArreglo(alto) de  $\alpha$   $\triangleright \mathcal{O}(\text{alto})$ 
7:     while j < alto do  $\triangleright \mathcal{O}(\text{copy}(\alpha))$  alto de veces  $\Rightarrow \mathcal{O}(\text{alto} \times \text{copy}(\alpha))$ 
8:       res[i][j]  $\leftarrow$  Copiar(a)  $\triangleright \mathcal{O}(\text{copy}(\alpha))$ 
9:       j  $\leftarrow$  j+1  $\triangleright \mathcal{O}(1)$ 
10:    i  $\leftarrow$  i+1  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(\text{ancho} \times \text{alto} \times \text{copy}(\alpha))$

Justificación: $\mathcal{O}(\text{ancho}) + \mathcal{O}(1) + \mathcal{O}(\text{ancho} \times \text{alto} \times \text{copy}(\alpha)) \Rightarrow \mathcal{O}(\text{ancho} \times \text{alto} \times \text{copy}(\alpha))$

Algorithm 94 iDefinir

```

1: procedure IDEFINIR(in/out e : estr, in ancho : nat, in alto : nat, in a :  $\alpha$ )
2:   (e.matriz[ancho])[alto]  $\leftarrow$  Copiar(a)  $\triangleright \mathcal{O}(1) \Rightarrow \mathcal{O}(\text{copy}(\alpha))$ 

```

Complejidad: $\mathcal{O}(\text{copy}(\alpha))$

Algorithm 95 $\bullet[\bullet][\bullet]$

```

1: procedure  $\bullet[\bullet][\bullet]$ (in/out e : estr, in ancho : nat, in alto : nat)  $\rightarrow$  res :  $\alpha$ 
2:   res  $\leftarrow$  (e.matriz[ancho])[alto]  $\triangleright \mathcal{O}(1) \Rightarrow \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$

Algorithm 96 iCopiar

```

1: procedure ICOPAR(in e : estr)  $\rightarrow$  res : estr
2:   res  $\leftarrow$  CrearMatriz(e.alto, e.ancho, e[0][0])  $\triangleright \mathcal{O}(\text{e.ancho} \times \text{e.alto} \times \text{copy}(\alpha))$ 
3:   i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
4:   while i < e.ancho do  $\triangleright \mathcal{O}(\text{alto} \times \text{copy}(\alpha))$  e.ancho de veces  $\Rightarrow \mathcal{O}(\text{e.ancho} \times \text{e.alto} \times \text{copy}(\alpha))$ 
5:     j  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
6:     while j < e.alto do  $\triangleright \mathcal{O}(\text{copy}(\alpha))$  e.alto de veces  $\Rightarrow \mathcal{O}(\text{e.alto} \times \text{copy}(\alpha))$ 
7:       Definir(res, i, j, Copiar(e.matriz[i][j]))  $\triangleright \mathcal{O}(1) + \mathcal{O}(\text{copy}(\alpha)) \Rightarrow \mathcal{O}(\text{copy}(\alpha))$ 
8:       j  $\leftarrow$  j+1  $\triangleright \mathcal{O}(1)$ 
9:       i  $\leftarrow$  i+1  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(\text{ancho} \times \text{alto} \times \text{copy}(\alpha))$

Justificacion: Llamo α a el tipo de elemento de e.matriz. $\mathcal{O}(\text{e.ancho} \times \text{e.alto} \times \text{copy}(\alpha)) + \mathcal{O}(1) + \mathcal{O}(\text{e.ancho} \times \text{e.alto} \times \text{copy}(\alpha)) = \mathcal{O}(\text{ancho} \times \text{alto} \times \text{copy}(\alpha))$.

Algorithm 97 iAlto

```

1: procedure IALTO(in e : estr)  $\rightarrow$  res : nat
2:   res  $\leftarrow$  e.alto  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$

Algorithm 98 iAncho

```

1: procedure IANCHO(in e : estr)  $\rightarrow$  res : nat
2:   res  $\leftarrow$  e.ancho  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$

Algorithm 99 iPosValida

```

1: procedure IPOVALIDA(in e : estr, in ancho : nat, in alto : nat)  $\rightarrow$  res : bool
2:   ancho < e.ancho  $\wedge$  alto < e.alto  $\triangleright \mathcal{O}(1)$ 

```

Complejidad: $\mathcal{O}(1)$

3.2. Matriz

TAD MATRIZ(α)

igualdad observacional

$$(\forall m, m' : \text{matriz}(\alpha)) \left(m =_{\text{obs}} m' \iff \left(\begin{array}{l} \text{ancho}(m) =_{\text{obs}} \text{ancho}(m') \wedge \text{alto}(m) =_{\text{obs}} \text{alto}(m') \\ \wedge_{\text{L}} ((\forall \text{col}, \text{fila} : \text{nat}) \text{posValida}(m, \text{col}, \text{fila}) \Rightarrow_{\text{L}}) \\ (\text{valor}(m, \text{col}, \text{fila}) =_{\text{obs}} \text{valor}(m', \text{col}, \text{fila}))) \end{array} \right) \right)$$

parámetros formales

géneros α

géneros $\text{matriz}(\alpha)$

exporta $\text{matriz}(\alpha)$, generadores, observadores, posValida

usa α , BOOL, NAT

observadores básicos

ancho	: $\text{matriz}(\alpha)$	\longrightarrow nat	
alto	: $\text{matriz}(\alpha)$	\longrightarrow nat	
valor	: $\text{matriz}(\alpha) \ m \times \text{columna} \ \text{nat} \times \text{fila} \ \text{nat}$	$\longrightarrow \alpha$	{posValida(m, columna, fila)}

generadores

crear	: $\text{ancho} \ \text{nat} \times \text{alto} \ \text{nat} \times \alpha$	$\longrightarrow \text{matriz}(\alpha)$	{ancho > 0 \wedge alto > 0}
definir	: $\text{columna} \ \text{nat} \times \text{fila} \ \text{nat} \times \alpha \times \text{matriz}(\alpha) \ m$	$\longrightarrow \text{matriz}(\alpha)$	{posValida(m, columna, fila)}

otras operaciones

posValida	: $\text{matriz}(\alpha) \times \text{columna} \times \text{fila}$	$\longrightarrow \text{bool}$
-----------	--	-------------------------------

axiomas $\forall m : \text{matriz}(\alpha), \forall an, al : \text{nat}, \forall a : \alpha$

ancho(crear(an, al, a))	$\equiv an$
ancho(definir(c, f, a, m))	$\equiv \text{ancho}(m)$
alto(crear(an, al, a))	$\equiv al$
alto(definir(c, f, a, m))	$\equiv \text{alto}(m)$
valor(crear(an, al, a), c, f)	$\equiv a$
valor(definir(c, f, a, m), c', f')	$\equiv \text{if } f = f' \wedge c = c' \text{ then } a \text{ else } \text{valor}(m, c', f') \text{ fi}$
posValida(m, c, f)	$\equiv f < \text{alto}(m) \wedge c < \text{ancho}(m)$

Fin TAD

3.3. Diccionario Acotado

TAD DICCIONARIOACOTADO(CLAVE, SIGNIFICADO)

igualdad observacional

$$(\forall d, d' : \text{diccAcot}(\kappa, \sigma)) \left(d =_{\text{obs}} d' \iff \left(\text{tamaño}(d) =_{\text{obs}} \text{tamaño}(d') \wedge \left((\forall c : \kappa) (\text{def?}(c, d) =_{\text{obs}} \text{def?}(c, d') \wedge_{\text{L}} (\text{def?}(c, d) \Rightarrow_{\text{L}} \text{obtener}(c, d) =_{\text{obs}} \text{obtener}(c, d'))) \right) \right) \right)$$

parámetros formales

géneros clave, significado

géneros diccAcot(clave, significado)

exporta diccAcot(clave, significado), generadores, observadores, borrar, claves

usa BOOL, NAT, CONJUNTO(CLAVE)

observadores básicos

def?	: clave \times diccAcot(clave, significado)	\longrightarrow bool	
obtener	: clave $c \times$ diccAcot(clave, significado) d	\longrightarrow significado	{def?(c, d)}
tamaño	: diccAcot(clave \times significado)	\longrightarrow nat	

generadores

vacío	: nat	\longrightarrow diccAcot(clave, significado)	
definir	: clave \times significado \times diccAcot(clave, significado) d	\longrightarrow diccAcot(clave, significado)	{#claves(d) < tamaño(d)}

otras operaciones

borrar	: clave $c \times$ diccAcot(clave, significado) d	\longrightarrow diccAcot(clave, significado)	{def?(c, d)}
claves	: diccAcot(clave, significado)	\longrightarrow conj(clave)	

axiomas $\forall d: \text{diccAcot}(\text{clave}, \text{significado}), \forall c, k: \text{clave}, \forall s: \text{significado}$

def?($c, \text{vacío}(n)$)	\equiv false
def?($c, \text{definir}(k, s, d)$)	$\equiv c = k \vee \text{def?}(c, d)$
obtener($c, \text{definir}(k, s, d)$)	\equiv if $c = k$ then s else obtener(c, d) fi
tamaño(vacío(n))	\equiv n
tamaño(definir(k, s, d))	\equiv tamaño(d)
borrar($c, \text{definir}(k, s, d)$)	\equiv if $c = k$ then if def?(c, d) then borrar(c, d) else d fi else definir($k, s, \text{borrar}(c, d)$) fi
claves(vacío(n))	$\equiv \emptyset$
claves(definir(c, s, d))	$\equiv \text{Ag}(c, \text{claves}(d))$

Fin TAD