Algoritmos y Estructuras de Datos II

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

Trabajo Práctico Nro. 2

Diseño - AED2-TEG

Grupo 11

Integrante	LU	Correo electrónico
Abásolo, Nicolás	310/08	nicolasabasolo@gmail.com
Menéndez, Emiliano	374/99	emystein@gmail.com
Méndez, Federico Martín	487/05	fmm_ab@hotmail.com
Otero, Guillermo	702/08	guillermo.otero@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

TP2: Diseño - AED2-TEG

Índice

1. Tablero

1.1. Interfaz

interfaz Tablero

parámetros formales

géneros μ

COPIAR(in m: μ)→ *res*: μ
{**Pre** ≡ *true*}
{**Post** ≡ $res =_{\text{obs}} m$}
Complejidad: O(copy(m))

se explica con Tablero(μ)

géneros tablero(μ)

Operaciones

INICIAL()→ *res*: **tablero**
{**Pre** ≡ *true*}
{**Post** ≡ $res = Inicial()$}
Complejidad: O(1)
Aliasing: no hay

CONTENER(inout t: `tablero(`μ`)`, in c: `casilla`, in m_i: μ, in m_v: μ)
{**Pre** ≡ $t = t_0 \land c \in Casillas(t) \land m_v \notin Salidas(t, c)$}
{**Post** ≡ $t = Contener(t_0, c, m_i, m_v)$}
Complejidad: O(f(ProxCasilla(t_0)) + copy(m_i) + copy(m_v))
Aliasing: no hay

AGREGAR(inout t: `tablero(`μ`)`, in c: `casilla`, in m_i: μ, in m_v: μ)
{**Pre** ≡ $t = t_0 \land c \in Casillas(t) \land m_v \notin Salidas(t, c)$}
{**Post** ≡ $t = Agregar(t_0, c, m_i, m_v)$}
Complejidad: O(f(ProxCasilla(t)) + f(ProxContinente(t)) + copy(m_i) + copy(m_v))
Aliasing: no hay

MOVILIZAR(inout t: `tablero(`μ`)`, in c_1: `casilla`, in c_2: `casilla`, in m: μ)
{**Pre** ≡ $t = t_0 \land \{c_1, c_2\} \subseteq Casillas(t) \land c_1 \neq c_2 \land \neg Conectadas(t, c_1, c_2) \land m \notin Salidas(t, c_1)$}
{**Post** ≡ $t = Movilizar(t_0, c_1, c_2, m)$}
Complejidad: O(copy(m))
Aliasing: no hay

CONECTAR(inout t: `tablero(`μ`)`, in c_1: `casilla`, in c_2: `casilla`, in m_i: μ, in m_v: μ)
{**Pre** ≡ $t = t_0 \land \{c_1, c_2\} \subseteq Casillas(t) \land c_1 \neq c_2 \land \neg Conectadas(t, c_1, c_2) \land m_i \notin Salidas(t, c_1) \land m_v \notin Salidas(t, c_2)$}
{**Post** ≡ $t = Conectar(t_0, c_1, c_2, m_i, m_v)$}
Complejidad: O(copy(m_i) + copy(m_v))
Aliasing: no hay

CASILLAS(in t: `tablero(`μ`)`)→ *res*: **conj(casilla)**
{**Pre** ≡ *true*}
{**Post** ≡ $res = Casillas(t)$}
Complejidad: O(1)
Aliasing: alias(*res*, Casillas(t))

CONTINENTE(in t: `tablero(`μ`)`, in c: `casilla`)→ *res*: **continente**
{**Pre** ≡ $c \in Casillas(t)$}
{**Post** ≡ $res = Continente(t, c)$}
Complejidad: O(1)
Aliasing: alias(*res*, Continente(t, c))

ORIGENES(in t: tablero(μ), in c: casilla, in m: μ)→ *res*: **conj(casilla)**
{**Pre** ≡ $c \in Casillas(t)$}
{**Post** ≡ $res = Origenes(t, c, m)$}
Complejidad: O(#Origenes(t, c, m))
Aliasing: no hay

CONTINENTES(in t: tablero(μ))→ *res*: **conj(continente)**
{**Pre** ≡ *true*}
{**Post** ≡ $res = Continentes(t)$}
Complejidad: O(1)
Aliasing: alias(*res*, Continentes(t))

CONECTADAS(in t: tablero(μ), in c_1: casilla, in c_2: casilla)→ *res*: **bool**
{**Pre** ≡ $\{c_1, c_2\} \subseteq Casillas(t)$}
{**Post** ≡ $res = Conectadas(t_0, c_1, c_2)$}
Complejidad: O(#Casillas(t))
Aliasing: no hay

SALIDAS(in t: tablero(μ), in c: casilla)→ *res*: **conj(μ)**
{**Pre** ≡ $c \in Casillas(t)$}
{**Post** ≡ $res = Salidas(t, c)$}
Complejidad: O(1)
Aliasing: alias(*res*, Salidas(t, c))

ORIGENESR(in t: tablero(μ), in c: casilla, in m: μ)→ *res*: **conj(casilla)**
{**Pre** ≡ $c \in Casillas(t)$}
{**Post** ≡ $res = OrigenesR(t, c, m)$}
Complejidad: O(#Origenes(t, c, m))
Aliasing: no hay

CASILLASDE(in t: tablero(μ), in cc: continente)→ *res*: **conj(casilla)**
{**Pre** ≡ $cc \in Continentes(t)$}
{**Post** ≡ $res = CasillasDe(t, cc)$}
Complejidad: O(1)
Aliasing: alias(*res*, CasillasDe(t, cc))

DESTINOS(in t: tablero(μ), in c: casilla)→ *res*: **itConj(conj(tupla(casilla, μ))**
{**Pre** ≡ $c \in Casillas(t)$}
{**Post** ≡ $res = CrearIt(Destinos(t, c))$}
Complejidad: O(1)
Aliasing: alias(*res*, Destinos(t, c))

PROXCASILLA(in t: tablero(μ))→ *res*: **casilla**
{**Pre** ≡ *true*}
{**Post** ≡ $res = \#Casillas(t) + 1$}
Complejidad: O(1)
Aliasing: no hay

PROXCONTINENTE(in t: tablero(μ))→ *res*: **continente**
{**Pre** ≡ *true*}
{**Post** ≡ $res = \#Continentes(t) + 1$}
Complejidad: O(1)
Aliasing: no hay

fin interfaz

1.2. Representación

1.2.1. Estructura de Representación

tablero(μ) se representa con **t**

donde **t** es tupla(casillas: dicc(casilla, datosCasilla), continentes: dicc(continente, conj(casilla)))

donde **datosCasilla** es tupla(salidas: conj(μ),
origenes: conj(tupla(casilla:casilla, movimiento:μ)),
destinos: conj(tupla(casilla:casilla, movimiento:μ)),
continente: continente)

1.2.2. Invariante de Representación

1. Para todo datosCasilla dc en t.datosCasilla, dc.continente está contenido en las claves de t.continentes.
2. Para todo conjunto de casillas cs en t.continentes, cs está incluído en claves de t.casillas.
3. Toda casilla en claves de t.casillas pertenece a uno solo de los conjuntos de casillas en t.continentes.
4. Para todo datosCasilla dc en t.datosCasilla, para toda tupla tup en dc.origenes: tup.casilla pertenece a t.casillas
5. Para todo datosCasilla dc en t.datosCasilla, para toda tupla tup en dc.destinos: tup.casilla pertenece a t.casillas
6. Para todo datosCasilla dc en t.datosCasilla, para toda tupla tupd en dc.destinos, tupd.movimiento pertenece a dc.salidas
7. Para todo datosCasilla dc en t.datosCasilla, Para todo movimiento m en t.salidas, existe una tupla en dc.destinos que tiene a m como segunda componente
8. Cada casilla c2 que aparece en las tuplas de destinos de c1, tiene una tupla en origenes donde la primer componente es c1 y la segunda componente es el mismo movimiento que el de destino de c1
9. Cada casilla c2 que aparece en las tuplas de origenes de c1, tiene una tupla en destinos donde la primer componente es c1 y la segunda componente es el mismo movimiento que el de origen de c1

Rep : t $\longrightarrow$ bool
Rep(t) $\equiv$ true $\Leftrightarrow$
($\forall$ c:casilla)(def?(c, t.casillas) $\Rightarrow$
($\forall$ dc:datosCasilla)(dc = Obtener(c, t.casillas) $\Rightarrow$
(dc.continente $\in$ claves(t.continentes)) **[1]** $\wedge$
(($\forall$ cc:continente)(def?(cc, t.continentes) $\Leftrightarrow$
(($\forall$ cs:conj(casilla))(cs = Obtener(cc, t.continentes) $\Leftrightarrow$ (cs $\in$ claves(t.casillas)) **[2]** $\wedge$ (($\exists!$ cs2:conj(casilla))(cs2 = cs $\wedge$ c $\in$ cs2)) **[3]**))))) $\wedge$
(($\forall$ tup:tupla(casilla, μ))((tup $\in$ dc.origenes $\Rightarrow$
tup.casilla $\in$ claves(t.casillas)) **[4]** $\wedge$
(tup $\in$ dc.destinos $\Rightarrow$ tup.casilla $\in$ claves(t.casillas) $\wedge$ tup.movimiento $\in$ dc.salidas) **[5 y 6]**)) $\wedge$
(($\forall$ ms:μ)(m $\in$ dc.salidas $\Rightarrow$ (($\exists$ tupd:tupla(casilla,μ))(tupd $\in$ dc.destinos $\wedge$ tupd.movimiento = m)))) **[7]** $\wedge$
(($\forall$ c2:casilla)(def?(c2, t.casillas) $\wedge$ c2 $\neq$ c $\Rightarrow$
(($\forall$ dc2:datosCasilla)(dc2 = Obtener(c2, t.casillas) $\Rightarrow$
(($\forall$ tup:tupla(casilla, μ))((tup $\in$ dc.destinos $\Rightarrow$
(($\exists$ tupo:tupla(casilla, μ))(tupo $\in$ dc2.origenes $\wedge$
tupo.casilla = c $\wedge$ tupo.movimiento = tup.movimiento))) **[8]** $\wedge$
(tup $\in$ dc.origenes $\Rightarrow$
(($\exists$ tupd:tupla(casilla, μ))(tupd $\in$ dc2.destinos $\wedge$
tupd.casilla = c $\wedge$ tupd.movimiento = tup.movimiento)) **[9]**
)
))
))
))
)
)

1.2.3. Función de Abstracción

Abs : e:t $\longrightarrow$ tablero(μ) {Rep(e)}

Abs(e) $\equiv$ t:tablero(μ) /
Casillas(t) = claves(e.casillas) $\wedge$
($\forall$ c:casilla)(c $\in$ Casillas(t) $\Rightarrow$
($\exists$ dc:datosCasilla)(dc = (Obtener(c, e.casillas)) $\wedge$
Continente(t, c) = dc.continente $\wedge$
($\forall$ m:μ)(Origenes(t, c, m) = OrigenesAux(dc.origenes, m))))
))

OrigenesAux : conj(tupla<casilla $\times$ μ>) os $\times$ μ m $\longrightarrow$ conj(casilla)
OrigenesAux(os, m) $\equiv$ **if**($\emptyset$?(os)) **then**
$\emptyset$
else
if(Π_2(damUno(os)) = m) **then**
Ag(Π_1(dameUno(os)), OrigenesAux(sinUno(os), m))
else
OrigenesAux(sinUno(os), m)
fi
fi

1.3. Algoritmos

iInicial()$\rightarrow$ *res* : *tablero*
1 **var** *continenteCero* : *continente*
2 *continenteCero* $\leftarrow$ 0
3 **var** *casillaCero* : *casilla*
4 *casillaCero* $\leftarrow$ 0
5 **var** *diccCasillas* : *dicc*(*casilla*, *datosCasilla*)
6 *diccCasillas* $\leftarrow$ *Vacio*()*//O(1)*
7 **var** *datosCasilla* : *datosCasilla*
8 *datosCasilla* $\leftarrow <$ *Vacio*(), *Vacio*(), *Vacio*(), *continenteCero* $>$*//O(1)*
9 *DefinirRapido*(*casillaCero*, *datosCasilla*, *diccCasillas*)*//O(1)*
10 **var** *diccContinentes* : *dicc*(*continente*, *conj*(*casilla*))
11 *diccContinentes* $\leftarrow$ *Vacio*()*//O(1)*
12 **var** *conjCasillas* : *conj*(*casilla*)
13 *conjCasillas* $\leftarrow$ *Vacio*()*//O(1)*
14 *AgregarRapido*(*conjCasillas*, *casillaCero*)*//O(1)*
15 *DefinirRapido*(*continenteCero*, *conjCasillas*, *diccContinentes*)*//O(#(conjCasillas)), en este caso hay una sola casilla*
16 *res* $\leftarrow <$ *diccCasillas*, *diccContinentes* $>$*//O(1) por referencia*

iContener(**inout** t : **tablero**(μ), **in** c : **casilla**, **in** m_i : μ, **in** m_v : μ)
1 **var** *datosCasillaExistente* : *datosCasilla*
2 *datosCasillaExistente* $\leftarrow$ *Obtener*(c, *t.casillas*)*//O(1) por referencia*
3 **var** *continente* : *continente*
4 *continente* $\leftarrow$ *datosCasillaExistente.continente//O(1) por referencia*
5 **var** *nuevaCasilla* : *casilla*
6 *nuevaCasilla* $\leftarrow$ *iProxCasilla*(*t*)*//O(1)*
7 **var** *datosNuevaCasilla* : *datosCasilla*
8 *//armo tupla de datos de la nueva casilla*
9 *datosNuevaCasilla* $\leftarrow <$ *AgregarRapido*($\emptyset$, *copy*(m_i)),*//O(copy(m_i))*
10 *AgregarRapido*($\emptyset$, $<$ c, *copy*(m_v $>$)),*//O(copy(m_v))*
11 *AgregarRapido*($\emptyset$, $<$ c, *copy*(m_i) $>$), *continente* $>$*//O(copy(m_i)*
12 *DefinirRapido*(*nuevaCasilla*, *datosNuevaCasilla*, *t.casillas*)*//O(copy(nuevaCasilla) + copy(datosNuevaCasilla))*
13 *//, en este caso es f(ProxCasilla(t)) por la expansion del vector en caso de que esté completo.*
14 **var** *conjCasillasContinente* : *conj*(*casilla*)
15 *conjCasillasContinente* $\leftarrow$ *Obtener*(*continente*, *t.continentes*)*//O(1) por referencia*
16 *AgregarRapido*(*conjCasillasContinente*, *nuevaCasilla*)*//O(1)//agrego casilla nueva como origen de c,m_i*
17 *AgregarRapido*(*datosCasillaExistente.origenes*, $<$ *nuevaCasilla*, *copy*(m_i) $>$)*//O(copy(m_i))*
18 *//agrego casilla nueva como destino de c,m_v*
19 *AgregarRapido*(*datosCasillaExistente.destinos*, $<$ *nuevaCasilla*, *copy*(m_v $>$))*//O(copy(m_v))*

iAgregar(inout t: tablero(μ), in c: casilla, in m_i: μ, in m_v: μ)

```
1  var datosCasillaExistente : datosCasilla
2  datosCasillaExistente ← Obtener(c, t.casillas)//O(1) por referencia
3  var continente : continente
4  continente ← datosCasillaExistente.continente//O(1) por referencia
5  var nuevoContinente : continente
6  nuevoContinente ← iProxContinente(t)//O(1)
7  var nuevaCasilla : casilla
8  nuevaCasilla ← iProxCasilla(t)//O(1)
9  var datosNuevaCasilla : datosCasilla
10 //armo tupla de datos de la nueva casilla
11 datosNuevaCasilla ←< AgregarRapido(∅, copy(m_i)),
12    AgregarRapido(∅, < c, copy(m_v) >),//O(copy(m_v))
13    AgregarRapido(∅, < c, copy(m_i) >), nuevoContinente >//O(copy(m_i) + copy(m_v))
14 DefinirRapido(nuevaCasilla, datosNuevaCasilla, t.casillas)//O(copy(m_i) + copy(m_v)) + f(ProxCasilla(t))
15 //, en este caso es f(ProxCasilla(t)) por la expansion del vector en caso de que esté completo.
16 var conjCasillasContinente : conj(casilla)
17 conjCasillasContinente ← AgregarRapido(nuevaCasilla, ∅)//O(1)
18 DefinirRapido(nuevoContinente, conjCasillasContinente, t.continentes)
19 //O(f(ProxContinente(t)) por la expansion del vector en caso de que esté completo.
20 //agrego casilla nueva como origen de c,m_i
21 AgregarRapido(datosCasillaExistente.origenes, < nuevaCasilla, copy(m_i) >)//O(copy(m_i))//agrego casilla nueva como desti
22 AgregarRapido(datosCasillaExistente.destinos, < nuevaCasilla, copy(m_v) >)//O(copy(m_v))
```

iMovilizar(inout t: tablero(μ), in c_1: casilla, in c_2: casilla, in m: μ)

```
1  var datosCasilla1 : datosCasilla
2  datosCasilla1 ← Obtener(c_1, t.casillas)//O(1) por referencia
3  AgregarRapido(datosCasilla1.salidas, m)//O(1)
4  AgregarRapido(datosCasilla1.destinos, < c_2, copy(m) >)//O(copy(m))
5  var datosCasilla2 : datosCasilla
6  datosCasilla2 ← Obtener(c_2, t.casillas)//O(1) por referencia
7  //agrego casilla nueva como origen de c_2,m
8  AgregarRapido(datosCasilla2.origenes, < c_1, copy(m) >)//O(copy(m))
```

iConectar(inout t: tablero(μ), in c_1: casilla, in c_2: casilla, in m_i: μ, in m_v: μ)

```
1  var datosCasilla1 : datosCasilla
2  datosCasilla1 ← Obtener(c_1, t.casillas)//O(1) por referencia
3  AgregarRapido(datosCasilla1.salidas, m_i)//O(1)
4  //agrego c_2 como origen de c_1,m_v
5  AgregarRapido(datosCasilla1.origenes, < c_2, copy(m_v) >)//O(copy(m_v))
6  AgregarRapido(datosCasilla1.destinos, < c_2, copy(m_i) >)//O(copy(m_i))
7  var datosCasilla2 : datosCasilla
8  datosCasilla2 ← Obtener(c_2, t.casillas)//O(1) por referencia
9  AgregarRapido(datosCasilla2.salidas, copy(m_v))//O(1)
10 //agrego casilla nueva como origen de c_2,m_i
11 AgregarRapido(datosCasilla2.origenes, < c_1, copy(m_i) >)//O(copy(m_i))
12 AgregarRapido(datosCasilla2.destinos, < c_1, copy(m_v) >)//O(copy(m_v))
```

iCasillas(in t: tablero(μ))→ res : conj(casilla)

```
1  res ← claves(t.casillas)//O(1) por referencia
2  return res
```

iContinente(in t: tablero(μ), in c: casilla)→ res : continente

```
1  var datosCasilla : datosCasilla
2  datosCasilla ← Obtener(c, t.casillas)//O(1) por referencia
3  res ← datosCasilla.continente//O(1) por referencia
4  return res
```

```
iOrigenes(in t: tablero(μ), in c: casilla, in m: μ)→ res : conj(casilla)
 1  res ← ∅
 2  var datosCasilla : datosCasilla
 3  datosCasilla ← Obtener(c, t.casillas)//O(1) por referencia
 4  var it : itConj
 5  it ← CrearIt(datosCasilla.origenes)//O(1) creo el iterador de conj
 6  var cm : tupla(casilla : casilla, movimiento : movimiento)
 7  while HaySiguiente(it)
 8        cm ← Siguiente(it)
 9        if cm.movimiento = m
10          then AgregarRapido(cm.casilla, res)
11               Avanzar(it)
12  return res
13  //recorro todas los origenes O(#Origenes(t, c, m))

iContinentes(in t: tablero(μ))→ res : conj(continente)
 1  res ← claves(t.continentes)//O(1) por referencia
 2  return res

iConectadas(in t: tablero(μ), in c1 : casilla, in c2 : casilla)→ res : bool
 1  //c2 es destino de c1?
 2  var datosCasilla : datosCasilla
 3  datosCasilla ← Obtener(c1, t.casillas)//O(1) por referencia
 4  var it : itConj
 5  it ← CrearIt(datosCasilla.destinos)//O(1) creo el iterador de conj
 6  var cm : tupla(casilla : casilla, movimiento : movimiento)
 7  res ← true
 8  while res ∧ HaySiguiente(it)
 9        cm ← Siguiente(it)
10        res ← (cm.casilla = c2)
11        Avanzar(it)
12        //recorro todos los destinos de c1 O(#Casillas(t))
13  //si c2 no es destino de c1, c1 es destino de c2?
14  if ¬res
15    then datosCasilla ← Obtener(c2, t.casillas)//O(1) por referencia
16         it ← CrearIt(datosCasilla.destinos)//O(1) creo el iterador de conj
17         res ← true
18         while res ∧ HaySiguiente(it)
19               cm ← Siguiente(it)
20               res ← cm.casilla = c1
21               Avanzar(it)//recorro todos los destinos de c2 O(#Casillas(t))
22  return res
23  //complejidad total: O(#Casillas(t))

iSalidas(in t: tablero(movimiento), in c: casilla)→ res : conj(movimiento)
 1  var datosCasilla : datosCasilla
 2  datosCasilla ← Obtener(c, t.casillas)//O(1) por referencia
 3  res ← datosCasilla.salidas//O(1) por referencia
 4  return res

iOrigenesR(in t: tablero(μ), in c: casilla, in m: μ)→ res : conj(casilla)
 1  res ← Vacio()//O(1)
 2  var it : itConj(conj(casilla))
 3  it ← crearIt(iOrigenes(t, c, m))
 4  var c′ : casilla
 5  while HaySiguiente(it)
 6        c′ ← Siguiente(it)
```

```
 7        AgregarRapido(res, c')
 8        Avanzar(it)
 9        //O(#Origenes(t, c, m))
10   if EsVacio?(res)
11     then AgregarRapido(res, c)
12   return res
13
```

iCasillasDe(in t: tablero(μ), in cc: continente)→ res : $conj(casilla)$
```
1  res ← Obtener(cc, t.continentes)//O(1) por referenciareturn res
```

iDestinos(in t: tablero(μ), in c: casilla)→ res : $itConj(conj(tupla(casilla, movimiento)))$
```
1  var datosCasilla : datosCasilla
2  datosCasilla ← Obtener(c, t.casillas)
3  var destinos : conj(tupla(casilla, movimiento))
4  destinos ← datosCasilla.destinos
5  res ← crearIt(destinos)//O(1) por referencia al primer elemento
6  return res
```

iProxCasilla(in t: tablero(movimiento))→ res : $casilla$
```
1  res ← #iCasillas(t)//O(1) tamaño del conj.
2  return res
```

iProxContinente(in t: tablero(movimiento))→ res : $continente$
```
1  res ← #iContinentes(t)//O(1) tamaño del conj.
2  return res
```

2. TEG

2.1. Interfaz

interfaz TEG

se explica con Tablero(μ)

géneros tablero(μ)

Operaciones

INICIAR(in t: `tablero(`μ`)`, in cs: `secu(casilla)`, in ms: `secu(mision)`)$\rightarrow$ res: **TEG**

{**Pre** $\equiv$ $cs \neq \emptyset \wedge long(cs) = long(ms) \wedge (\forall c : casilla)(Esta?(c, cs) \Rightarrow c \in Casillas(t)) \wedge (\forall m : mision)(Esta?(m, ms) \Rightarrow m \in Continentes(t)) \wedge SinRepetidos(cs)$}

{**Post** $\equiv res = Iniciar(t, cs, ms)$}

Complejidad: O(FALTA)

Aliasing: FALTA

fin interfaz

2.2. Representación

2.2.1. Estructura de Representación

tablero(μ) se representa con t
donde t es tupla
<
  tablero: datosTablero,
  paises: dicc(casilla, datosCasilla),
  jugadores: dicc(jugador, datosJugador),
  jugadoresJugando: conj(jugador),
  casillasDisputadas: conj(casilla)
>

datosTablero es ad(tupla < conj(tupla < μ, casilla >), continente >)

datosCasilla es tupla
<
  disputan: conjA(jugador),
  fichas: mconj(jugador),
  disputantes: heap(nodo)
>

datosJugador es tupla
<
  fichasHistoria: nat,
  mision: continente,
  fichasEnJuego: nat,
  porDominar: nat,
  dominadas: conj(casilla)
>

nodo es tupla
<
  rep: nat,
  jugador: jugador,
  izq: *nodo,
  der: *nodo
>

2.2.2. Invariante de Representación

Rep : t $\longrightarrow$ bool
Rep(t) $\equiv$ true $\Leftrightarrow$

2.2.3. Función de Abstracción

Abs : t $\longrightarrow$ TEG $\{\text{Rep(t)}\}$
Abs(t) $\equiv$

2.3. Algoritmos

iINICIAR(in t: tablero(μ), in cs: secu(casilla), in ms: secu(mision)) $\rightarrow$ $res : TEG$

```
1  var itCasillas : itSecu
2  var itMisiones : itSecu
3  var diccCasillas : dicc(casilla, datosCasilla)
4  diccCasillas ← Vacio()//O(1)
5  var diccJugadores : dicc(jugador, datosJugador)
6  diccJugadores ← Vacio()//O(1)
7  var conjJugadoresJugando : conj(jugador)
8  conjJugadoresJugando ← Vacio()//O(1)
9  var casillaActual : casilla
10 var datosCasillaActual : datosCasilla
11 var jugadorActual : jugador
12 jugadorActual ← 0
13 var datosJugadorActual : datosJugador
14 var disputan : conjA(jugador)
15 var fichas : mconj(jugador)
16 var disputantes : heap(nodo)
17 var nodoActual : nodo
18 //copiar tablero (origenes y continente de cada casilla)
19 itCasillas ← CrearIt(cs)//O(1) creo el iterador de conj
20 itMisiones ← CrearIt(ms)//O(1) creo el iterador de conj
21 while HaySiguiente(itCasillas)
22       casillaActual ← Siguiente(itCasillas)
23       misionActual ← Siguiente(itMisiones)
24       disputan ← Vacio()//O(long(cs))
25       fichas ← Vacio()//O(long(cs)) mconj sobre arreglo indexado por jugador, long(cs) = cant. de jugadores
26       AgregarRapido(fichas, jugadorActual)//O(1)
27       disputantes ← Vacio()//O(long(cs)) creo heap vacio sobre un arreglo con long. cant. de jugadores
28       datosCasillaActual ←< disputan, fichas, disputantes >//O(1)
29       DefinirRapido(casillaActual, datosCasillaActual, diccCasillas)//O(1)
30       //armo datosJugadorActualdatosJugadorActual ← iCrearDatosJugador(t, jugadorActual, casillaActual, misionActual)//O(1)
31       DefinirRapido(jugadorActual, datosJugadorActual, diccJugadores)//O(1)
32       AgregarRapido(conjJugadoresJugando, jugadorActual)//O(1)
33       jugadorActual ← jugadorActual + 1
34       Avanzar(itCasillas)
35       Avanzar(itMisiones)
36
37 res.tablero ← iCopiarTablero(t)
38 res.paises ← diccCasillas//O(1) por referencia
39 res.jugadores ← diccJugadores//O(1) por referencia
40 res.jugadoresJugando ← conjJugadoresJugando//O(1) por referencia
41 res.casillasDisputadas ← Vacio()//O(1)
42 return res
```

iCREARDATOSJUGADOR(in t: tablero(μ), in j: jugador, in c: casilla, in m: continente) $\rightarrow$ $res : datosJugador$

```
1 var porDominar : nat
2 porDominar ← iPorDominarTablero(t, j, c, m)//O(1)
```

```
3  var cs : conj(casilla)
4  cs ← AgregarRapido(Vacio(), c)//O(1)
5  res ←< 1, m, 1, porDominar, cs >//O(1) por referencia
6  return res
```

iPorDominarTablero(in t: tablero(μ), in j: jugador, in c: casilla, in m: continente)→ res : nat

```
1  var cantPaisesContinente : nat
2  cantPaisesContinente ← #CasillasDe(t, m)//cardinal de conj. lineal es O(1)
3  res ← cantPaisesContinente
4  //Continente es O(1)
5  if Continente(t, c) = m
6    then
7          res ← res − 1
8  return res
9  //complejidad total O(1)
```

iCopiarTablero(in t: tablero(μ))→ res : $ad(< conj(< casilla, movimiento >), continente >)$

```
1  res ← CrearArreglo(#(Casillas(t)))//O(#(Casillas(t)))
2  var itCasillas : itConj
3  var casillaActual : casilla
4  var itDestinos : itConj
5  var cm : tupla < casilla, μ >
6  var casillaDestino : casilla
7  var m : μ
8  //defino el continente y defino un conj. vacío en las tuplas ¡conj(casilla,movimiento),continente¿ de res
9  itCasillas ← CrearIt(Casillas(t))//O(1) creo el iterador de conj
10 while HaySiguiente(itCasillas)
11       casillaActual ← Siguiente(itCasillas)
12       res[casillaActual] ←< Vacio(), Continente(t, casillaActual) >//Continente(t,c) tiene comlejidad O(1)
13       Avanzar(itCasillas)
14       //complejidad del ciclo O(#Casillas(t))
15 //lleno cada posición de res con origenes
16 itCasillas ← CrearIt(Casillas(t))//O(1) creo el iterador de conj
17 while HaySiguiente(itCasillas)
18       casillaActual ← Siguiente(itCasillas)
19       itDestinos ← Destinos(t, casillaActual)//O(1)
20       //para cada destino casillaDestino,m de casillaActual, agrego casillaActual,m a origenes de casillaDestino
21       while HaySiguiente(itDestinos)
22             cm ← Actual(itDestinos)
23             casillaDestino ← Π1(cm)
24             m ← Π2(cm)
25             datosCasillaDestino ← res[casillaDestino]//O(1) por referencia
26             conjOrigenes ← Π1(datosCasillaDestino)//O(1) por referencia
27             AgregarRapido(conjOrigenes, < casillaActual, m >)
28             Avanzar(itDestinos)
29       //complejidad del ciclo O(#Destinos(t, casillaActual))
30       Avanzar(itCasillas)//complejidad del ciclo O(#Casillas(t))
31 return res
```

iJugadores(in t: TEG)→ res : $conj(jugador)$

```
1  return claves(t.jugadores)
```

iFichas(in t: TEG, in c: casilla)→ res : $mconj(jugador)$

```
1  return (obtener(c, t.paises)).fichas
```

iMision(in t: TEG, in j: jugador)→ res : $continente$

```
1  return (obtener(j, t.jugadores)).mision
```

iFichasAgregadas(in t: TEG, in j: jugador)→ res : nat
1 $return$ $(obtener(j, t.jugadores)).fichasHistoria$

iDominada?(in t: TEG, in c: casilla)→ res : $bool$
1 $return$ $(\#((obtener(c, t.paises)).disputan) = 1)$

iVacia?(in t: TEG, in c: casilla)→ res : $bool$
1 $return$ $(\#((obtener(c, t.paises)).disputan) = 0)$

iDisputada?(in t: TEG, in c: casilla)→ res : $bool$
1 $return$ $(\#((obtener(c, t.paises)).disputan) > 0)$

iDisputan(in t: TEG, in c: casilla)→ res : $conj(jugador)$
1 $return$ $(obtener(c, t.paises)).disputan$

iDominador(in t: TEG, in c: casilla)→ res : $jugador$
1 $return$ $dameUno((obtener(c, t.paises)).fichas)$

iDominadas(in t: TEG, in j: jugador)→ res : $conj(casilla)$
1 $return$ $(obtener(j, t.jugadores)).dominadas$

iPorDominar(in t: TEG, in j: jugador)→ res : nat
1 $return$ $(obtener(j, t.jugadores)).porDominar$

iMisionCumplida?(in t: TEG, in j: jugador)→ res : $bool$
1 $return$ $((obtener(j, t.jugadores)).porDominar = 0)$

iTieneFichas?(in t: TEG, in j: jugador)→ res : $bool$
1 $return$ $((obtener(j, t.jugadores)).fichasEnJuego > 0)$

iConFichas(in t: TEG)→ res : $conj(jugador)$
1 $return$ $t.jugadoresJugando$

iEliminados(in t: TEG)→ res : $conj(jugador)$
1 $return$ $(claves(t.jugadores)\ t.jugadoresJugando)$

iCumplieronMision(in t: TEG)→ res : $conj(jugador)$
1 $return$ $(claves(t.jugadores)\ t.jugadoresJugando\ eliminados(t))$

iGanadores(in t: TEG)→ res : $conj(jugador)$
1 **var** res : $conj(jugador)$
2 **if** $(\#(conFichas(t)) = 1)$
3 **then** $conFichas(t)$
4 **else** $cumplieronMision(t)$
5 **fi**
6 $return$ res

iFinalizado?(in t: TEG)→ res : $bool$
1 $return$ $(ganadores(t) \neq \emptyset)$

iCasillas(in t: TEG)→ res : $conj(casilla)$
1 $return$ $claves(t.paises)$

iPuedeAgregar?(in t: TEG, in j: jugador, in c: casilla)→ res : $bool$
1 $return$ $(\neg Disputada(t, c) \wedge Pertenece(t.jugadoresJugando, j) \wedge \neg Finalizado?(t))$

iContinentes(in t: TEG)→ res : $conj(continente)$
1 **var** i : int

2 $i \leftarrow 0$
3 **var** $res : conj(continente)$
4 **while** $i < long(tablero)$
5 $\quad Ag(res, \Pi_2(tablero[i]))$
6 *return* res