

# Algoritmos y Estructuras de Datos II

## Trabajo Práctico 3

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

### Pacalgo2

Los inertes

Integrante	LU	Correo electrónico
Bruno Robbio	480/09	brobbio@hotmail.com
Nicolas Andres Kinaschuk	248/15	nicolaskinaschuk@gmail.com
Pedro Joel Burgos	804/18	facultadburgospedrojoel@hotmail.com
Valentina Madelaine Saravia Ruiz	257/18	valentina.saraviaruiz@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

# Modulo Mapa

## Interfaz

se explica con: MAPA

géneros: mapa.

### Operaciones básicas de mapa

**NUEVOMAPA**(*in largo: nat, in alto: nat, in inicio: coordenada, in llegada: coordenada, in fantasmas: conj(coordenada), in paredes: conj(coordenada), in chocolates: conj(coordenada)*)  $\rightarrow res : \text{mapa}$   
**Pre**  $\equiv \{(inicio \neq llegada \wedge todosEnRango(paredes \cup fantasmas \cup chocolates \cup \{inicio, llegada\}, largo, alto) \wedge \{inicio, llegada\} \cap (fantasmas \cup paredes) = \emptyset \wedge disjuntosDeAPares(paredes, fantasmas, chocolates))\}$   
**Post**  $\equiv \{res = nuevoMapa(largo, alto, inicio, llegada, paredes, fantasmas, chocolates) \}$   
**Complejidad:**  $O(alto \cdot largo \cdot (\#chocolates + \#fantasmas + \#paredes + 1))$   
**Descripción:** Genera un nuevo mapa  
**Aliasing:** Para construir el mapa hacemos copia de todos los conjuntos

**ESCASILLEROPELIGROSO**(*in m: mapa, in posicion: coordenada*)  $\rightarrow res : \text{bool}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res = distConFantasmasMasCercano(fantasmas(m), posicion) \leq 3 \}$   
**Complejidad:**  $O(1)$   
**Descripción:** Devuelve true si el casillero es peligroso, es peligroso si existe un fantasma con distancia  $\leq 3$  respecto a la posición

**ENRANGO**(*in m: mapa, in posicion: coordenada*)  $\rightarrow res : \text{bool}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res = enRango(posicion, largo(m), alto(m))\}$   
**Complejidad:**  $O(1)$   
**Descripción:** Devuelve true si la posicion se encuentra en rango

**CANTCHOCOLATES**(*in map: mapa*)  $\rightarrow res : \text{nat}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res = \#(chocolates(map))\}$   
**Complejidad:**  $O(c)$   
**Descripción:** Devuelve la cantidad de chocolates en el mapa

**ESPARED**(*in map: mapa, in posicion: coordenada*)  $\rightarrow res : \text{bool}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res = \text{true} \iff posicion \in paredes(map)\}$   
**Complejidad:**  $O(1)$   
**Descripción:** Devuelve el conjunto de paredes

**INICIO**(*in map: mapa*)  $\rightarrow res : \text{coordenada}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res = inicio(map)\}$   
**Complejidad:**  $O(1)$   
**Descripción:** Devuelve la coordenada de inicio del mapa

**LLEGADA**(*in map: mapa*)  $\rightarrow res : \text{coordenada}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res = llegada(map)\}$   
**Complejidad:**  $O(1)$   
**Descripción:** Devuelve la coordenada de llegada del mapa

**IDCHOCOLATE**(**in**  $m$  : mapa, **in**  $posicion$  : coordenada)  $\rightarrow res$  : int  
**Pre**  $\equiv \{enRango(posicion)\}$   
**Post**  $\equiv \{posicion \in chocolates(m) \iff 0 \leq res < \#chocolates(m)\}$   
**Complejidad:**  $O(1)$   
**Descripción:** Devuelve el id del chocolate en el mapa

## Implementación

### Representación

mapa se representa con *mp*

donde *casillero* es tupla(*fantasma*: bool,  
  *peligrosa*: bool,  
  *pared*: bool,  
  *idChocolate*: int)

donde *columna* es array[0...*largo*] de *casillero*

donde *mp* es tupla(*matriz*: array[0...*alto*] de *columna*,  
  *#chocolates*: nat,  
  *alto*: nat,  
  *largo*: nat,  
  *inicio*: coordenada,  
  *llegada*: coordenada)

### Invariante de representación

Rep : mp  $\longrightarrow$  boolean

Rep(*e*)  $\equiv$  True  $\iff$  ( $0 \leq e.inicio_1 < e.largo \wedge 0 \leq e.inicio_2 < e.alto$ )  $\wedge$   
  ( $0 \leq e.llegada_1 < e.largo \wedge 0 \leq e.llegada_2 < e.alto$ )  $\wedge$   
  ( $e.inicio \neq e.llegada$ )  $\wedge$   
   $(\forall i: \text{nat})(0 \leq i < e.largo) \Rightarrow_L$  ( $(\forall j: \text{nat})(0 \leq j < e.alto) \Rightarrow_L$  ( $(\beta(e.matriz[i][j].pared) +$   
     $\beta(e.matriz[i][j].fantasma) +$   
     $\beta(0 \leq e.matriz[i][j].idChocolate < e.\#chocolates) \leq 1) \wedge$   
   $((\forall n: \mathbb{N})(0 \leq n < e.\#chocolates) \Rightarrow$   
     $(\exists! i, j: \text{nat})(0 \leq i < e.largo \wedge 0 \leq j < e.alto) \wedge_L$   
       $(e.matriz[i][j].chocolates = n)) \wedge$   
   $(e.matriz[i][j].peligrosa \Rightarrow_L (\exists n, m: \text{nat})(0 \leq n < e.largo \wedge 0 \leq m < e.alto) \wedge_L$   
     $(e.matriz[n][m].fantasma \wedge \text{distancia}(\langle i, j \rangle, \langle n, m \rangle) \leq 3)))$   
  )

## Función de abstracción

$\text{Abs} : \text{mp } e \longrightarrow \text{Mapa}$   $\{\text{Rep}(e)\}$   
 $(\forall e : \text{mp}) \text{ Abs}(e) =_{\text{obs}} m : \text{mapa} \mid \text{largo}(m) = e.\text{largo} \wedge$   
 $\text{alto}(m) = e.\text{alto} \wedge$   
 $\#(\text{chocolates}(m)) = e.\#\text{chocolates} \wedge$   
 $e.\text{inicio} = \text{inicio}(m) \wedge$   
 $e.\text{llegada} = \text{llegada}(m) \wedge$   
 $(\forall i : \text{nat})(0 \leq i < e.\text{largo}) \Rightarrow_L ($   
 $(\forall j : \text{nat})(0 \leq j < e.\text{alto}) \Rightarrow_L ($   
 $e.\text{matriz}[i][j].\text{fantasma} \iff \langle i, j \rangle \in \text{fantasmas}(m) \wedge$   
 $e.\text{matriz}[i][j].\text{paredes} \iff \langle i, j \rangle \in \text{paredes}(m) \wedge$   
 $0 \leq e.\text{matriz}[i][j].\text{idChocolate} < e.\#\text{chocolates} \iff \langle i, j \rangle \in \text{chocolates}(m) \wedge$   
 $e.\text{matriz}[i][j].\text{peligrosa} \iff \text{distConFantasmasMasCercano}(\text{fantasmas}(m), \langle i, j \rangle) \leq 3))$

## Algoritmos

---

**iNuevoMapa**(in *largo*: nat, in *alto*: nat, in *inicio*: coordenada, in *llegada*: coordenada, in *fantasmas*:  
 conj (coordenada), in *paredes*: conj (coordenada), in *chocolates*: conj (coordenada))  $\rightarrow$  *res* : mapa

1:	int contadorDeChocolate $\leftarrow$ 0	$\triangleright \mathcal{O}(1)$
2:	mapa $\leftarrow \langle \rangle$	$\triangleright \mathcal{O}(1)$
3:	mapa.inicio $\leftarrow$ inicio	$\triangleright \mathcal{O}(1)$
4:	mapa.llegada $\leftarrow$ llegada	$\triangleright \mathcal{O}(1)$
5:	mapa.largo $\leftarrow$ largo	$\triangleright \mathcal{O}(1)$
6:	mapa.alto $\leftarrow$ alto	$\triangleright \mathcal{O}(1)$
7:	mapa.#chocolates $\leftarrow$ chocolates.longitud	$\triangleright \mathcal{O}(1)$
8:	<b>para</b> int $x = 0; x < \text{mapa.largo}; x++$ <b>hacer</b>	$\triangleright \mathcal{O}(\text{largo})$
9:	<b>para</b> int $y = 0; y < \text{mapa.alto}; y++$ <b>hacer</b>	$\triangleright \mathcal{O}(\text{largo} * \text{alto})$
10:	mapa.matriz[x][y].pared = (x, y) in paredes	$\triangleright \mathcal{O}(\text{largo} * \text{alto} *  P )$
11:	mapa.matriz[x][y].fantasma = (x, y) in fantasmas	$\triangleright \mathcal{O}(\text{largo} * \text{alto} *  F )$
12:	mapa.matriz[x][y].peligrosa = seAsusta((x, y), fantasmas)	$\triangleright \mathcal{O}(\text{largo} * \text{alto} *  F )$
13:	<b>si</b> (x, y) in chocolates <b>entonces</b>	$\triangleright \mathcal{O}(\text{largo} * \text{alto} *  C )$
14:	mapa.matriz[x][y].idChocolate $\leftarrow$ contadorDeChocolate	$\triangleright \mathcal{O}(\text{largo} * \text{alto} *  C )$
15:	contadorDeChocolate $++$	$\triangleright \mathcal{O}(\text{largo} * \text{alto} *  C )$
16:	<b>else</b>	
17:	mapa.matriz[x][y].idChocolate $\leftarrow -1$	$\triangleright \mathcal{O}(\text{largo} * \text{alto} *  C )$
18:	res $\leftarrow$ mapa	$\triangleright \mathcal{O}(1)$

Complejidad:  $\mathcal{O}((\text{alto} \cdot \text{largo} \cdot (\#\text{chocolates} + \#\text{fantasmas} + \#\text{paredes} + 1)))$

---

---

**seAsusta**(in  $c$ : coordenada, in  $fantasmas$ : conj(coordenada) out  $res$ : bool)

```

1:  $res \leftarrow False$ 
2:  $int\ i \leftarrow 0$ 
3: mientras  $i < fantasma.longitud$  hacer
4:   si  $distancia(c, fantasma[i]) \leq 3$  entonces
5:      $res \leftarrow true$ 
6:   else
7:      $i++$ 

```

$\triangleright \mathcal{O}(1)$   
 $\triangleright \mathcal{O}(|f|)$   
 $\triangleright \mathcal{O}(|f|)$   
 $\triangleright$  si es el peor caso no entra  
 $\triangleright \mathcal{O}(|f|)$

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

---



---

**distancia**(in  $c$ : coordenada, in  $f$ : coordenada out  $res$ : nat)

```

1:  $res \leftarrow valorAbsoluto(c_1 - f_1) + valorAbsoluto(c_2 - f_2)$ 

```

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

**iesCasilleroPeligroso**(in  $m$ : mapa, in  $posicion$ : coordenada)  $\rightarrow res$ : bool

```

1: si  $enRango(m, posicion) \wedge_L m.matriz.[posicion_1][posicion_2].peligrosa$  entonces
2:    $res \leftarrow true$ 
3: else
4:    $res \leftarrow false$ 

```

$\triangleright \Theta(1)$   
 $\triangleright \Theta(1)$   
 $\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

**ienRango**(in  $m$ : mapa, in  $posicion$ : coordenada)  $\rightarrow res$ : bool

```

1:  $res \leftarrow (0 \leq posicion_1 < m.alto \wedge_L 0 \leq posicion_2 < m.largo)$ 

```

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

**iCantChocolates**(in  $map$ : mapa)  $\rightarrow res$ : nat

```

1:  $res \leftarrow map.\#chocolates$ 

```

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

**iEsPared**(in  $map$ : mapa, in  $posicion$ : coordenada)  $\rightarrow res$ : bool

```

1:  $res \leftarrow map.matriz.[posicion_1][posicion_2].pared$ 

```

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

**iInicio**(in  $map$ : mapa)  $\rightarrow res$ : coordenada

```

1:  $res \leftarrow map.inicio$ 

```

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

**iLlegada**(in  $map$ : mapa)  $\rightarrow res$ : coordenada

```

1:  $res \leftarrow map.llegada$ 

```

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---

---



---

**idChocolate**(in  $m$ : mapa, in  $posicion$ : coordenada)  $\rightarrow res$  :  $int$

1:  $res \leftarrow m.matriz.[posicion_1][posicion_2].idChocolate$   $\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---

## Servicios usados

-

# Modulo Partida

## Interfaz

se explica con: PARTIDA

géneros: partida.

### Operaciones básicas de partida

NUEVAPARTIDA(**in**  $m$ : mapa)  $\rightarrow res$  : partida

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

**Post**  $\equiv \{res = \text{nuevaPartida}(m)\}$

**Complejidad:**  $O(c)$ ,  $c$  es la cantidad de chocolates que contiene el mapa

**Descripción:** Genera una nueva partida

**Aliasing:** El mapa se recibe por referencia

MOVER(**in/out**  $p$ : partida, **in**  $d$ : dirección)

**Pre**  $\equiv \{p_0 = p\}$

**Post**  $\equiv \{p = \text{mover}(p_0, d)\}$

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

**Descripción:** Mueva la posición del jugador un casillero

**Aliasing:** Se modifica  $p$  internamente

GANÓ?(**in**  $p$ : partida)  $\rightarrow res$  : bool

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

**Post**  $\equiv \{res = \text{ganó?}(p)\}$

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

**Descripción:** Devuelve true si el jugador gana la partida

PERDIÓ?(**in**  $p$ : partida)  $\rightarrow res$  : bool

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

**Post**  $\equiv \{res = \text{perdió?}(p)\}$

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

**Descripción:** Devuelve true si el jugador perdio la partida

JUGADOR(**in**  $p$ : partida)  $\rightarrow res$  : coordenada

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

**Post**  $\equiv \{res = \text{perdió?}(p)\}$

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

**Descripción:** Devuelve la posicion del jugador

CANTMOV(**in**  $p$ : partida)  $\rightarrow res$  : nat

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

**Post**  $\equiv \{res = \text{cantMov}(p)\}$

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

**Descripción:** Devuelve la cantidad de movimientos del jugador



## Implementación

### Representación

partida se representa con  $pt$

donde  $pt$  es tupla( $mapa$ : mp,  
   $jugador$ : coordenada,  
   $chocolates$ : array[0...c] de bool,  
   $cantMov$ : nat,  
   $inmunidad$ : nat,  
   $gano$ : bool,  
   $perdio$ : bool)

### Funciones auxiliares

$distancia : coordenada \times coordenada \longrightarrow nat$

$distanciaMinima : coordenada \times conj(coordenada) \longrightarrow bool$

$chocolatesSinComer : pt \longrightarrow conj(coordenada)$

$distancia(x, y) \equiv |x_1 - y_1| + |x_2 - y_2|$

$distanciaMinima(j, c) \equiv \text{if } \#(c) = 1 \text{ then}$   
   $distancia(j, dameUno(c))$   
  **else**  
     $\text{mín}(distancia(j, dameUno(c)), distanciaMinima(j, sinUno(c)))$   
  **fi**

$chocolatesSinComer(e) \equiv$  Esta función devuelve el conjunto de coordenadas de chocolates en la  $e.mapa.matriz$  cuyo Id en el array de la partida ( $e.chocolates$ ) aun estan en true

### Invariante de representación

$Rep : pt \longrightarrow boolean$

$Rep(e) \equiv \text{True} \iff (mapa.EnRango(e.mapa, e.jugador) \wedge$   
   $long(e.chocolates) = e.mapa.\#chocolates) \wedge_L$   
   $e.cantMov = 0 \Rightarrow$   
     $e.mapa.inicio = jugador \wedge$   
    **if**  $0 \leq e.mapa.matriz[jugador_1][jugador_2].idChocolate < e.mapa.\#chocolates$  **then**  
       $e.chocolates[e.mapa.matriz[e.jugador_1][e.jugador_2].idChocolate] = false \wedge$   
       $e.inmunidad = 10 \wedge$   
       $(\forall i : \mathbb{N})(0 \leq i < mapa.\#chocolates \wedge i \neq e.mapa.matriz[e.jugador_1][e.jugador_2].idChocolate)$   
       $\Rightarrow_L (e.chocolates[i] = true)$   
    **else**  
       $e.inmunidad = 0 \wedge (\forall i : \mathbb{N})(0 \leq i < mapa.\#chocolates) \Rightarrow_L (e.chocolates[i] = true)$   
    **fi**  $\wedge$   
     $(e.chocolates[e.mapa.matriz[e.jugador_1][e.jugador_2].idChocolate] = false) \wedge$   
     $e.inmunidad=10 \Rightarrow 0 \leq e.mapa.matriz[e.jugador_1][e.jugador_2].idChocolate < e.mapa.\#chocolates \wedge$   
     $e.inmunidad \leq 10 - distanciaMinima(jugador, chocolatesSinComer(e)) \wedge$   
     $e.gano \iff jugador = e.mapa.llegada \wedge$   
     $e.perdio \iff e.inmunidad = 0 \wedge e.mapa.distConFantasmasMásCercano(e.mapa, e, jugador) \leq 3$

### Función de abstracción

Abs : pt  $e \rightarrow$  partida

{Rep( $e$ )}

$(\forall e : \text{pt}) \text{ Abs}(e) =_{\text{obs}} p : \text{partida} \mid \text{mapa}(p) = e.\text{mapa} \wedge$   
 $\text{jugador}(p) = e.\text{jugador} \wedge$   
 $\text{chocolates}(p) = \text{chocolatesSinComer}(e) \wedge$   
 $\text{cantMov}(p) = e.\text{cantMov} \wedge$   
 $\text{inmunidad}(p) = e.\text{inmunidad} \wedge$   
 $\text{ganó?}(p) = e.\text{gano} \wedge$   
 $\text{perdió?}(p) = e.\text{perdio}$

## Algoritmos

---

**iNuevaPartida**(in  $m : \text{mapa}$ )  $\rightarrow res : \text{partida}$

1: $partida \leftarrow \langle \rangle$	$\triangleright \mathcal{O}(1)$
2: $partida.\text{mapa} \leftarrow m$	$\triangleright \mathcal{O}(1)$
3: $partida.\text{jugador} \leftarrow \text{mapa.inicio}(m)$	$\triangleright \mathcal{O}(1)$
4: $i \leftarrow 0$	$\triangleright \mathcal{O}(1)$
5: <b>mientras</b> $i < \text{mapa.cantChocolates}(m)$ <b>hacer</b>	$\triangleright \mathcal{O}(c)$
6: $partida.\text{chocolates}[i] \leftarrow \text{true}$	$\triangleright \mathcal{O}(c)$
7: $i++$	$\triangleright \mathcal{O}(c)$
8: $partida.\text{cantMov} \leftarrow 0$	$\triangleright \mathcal{O}(1)$
9: $\text{idChocolate} \leftarrow \text{mapa.idChocolate}(m, partida.\text{jugador})$	$\triangleright \mathcal{O}(1)$
10: $\text{hayUnChocolate} \leftarrow 0 \leq \text{idChocolate} < \text{mapa.cantChocolates}(m)$	$\triangleright \mathcal{O}(1)$
11: <b>si</b> $\text{hayUnChocolate}$ <b>entonces</b>	$\triangleright \mathcal{O}(1)$
12: $partida.\text{chocolates}[\text{idChocolate}] \leftarrow \text{false}$	$\triangleright \mathcal{O}(1)$
13: $partida.\text{inmunidad} \leftarrow 10$	$\triangleright \mathcal{O}(1)$
14: <b>else</b>	
15: $partida.\text{inmunidad} \leftarrow 0$	$\triangleright \mathcal{O}(1)$
16: $partida.\text{perdio} \leftarrow \text{mapa.esCasilleroPeligroso}(m, partida.\text{jugador}) \wedge partida.\text{inmunidad} = 0$	$\triangleright \mathcal{O}(1)$
17: $partida.\text{gano} \leftarrow \text{false}$	$\triangleright \mathcal{O}(1)$
18: $res \leftarrow partida$	$\triangleright \mathcal{O}(1)$

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

Justificación: La complejidad de este algoritmo es igual a recorrer el array  $partida.\text{chocolates}$ , que tiene longitud  $c$ , que es igual a la cantidad de chocolates que contiene el mapa, para inicializar sus valores en  $\text{true}$

---

---



---

**iMover**(in/out  $p$ : partida, in  $d$ : direccion)

1:	<b>si</b> $d = \text{ARRIBA}$ <b>entonces</b>	$\triangleright \mathcal{O}(1)$
2:	$\text{desplazamiento} \leftarrow (0, 1)$	$\triangleright \mathcal{O}(1)$
3:	<b>si</b> $d = \text{ABAJO}$ <b>entonces</b>	$\triangleright \mathcal{O}(1)$
4:	$\text{desplazamiento} \leftarrow (0, -1)$	$\triangleright \mathcal{O}(1)$
5:	<b>si</b> $d = \text{IZQUIERDA}$ <b>entonces</b>	$\triangleright \mathcal{O}(1)$
6:	$\text{desplazamiento} \leftarrow (-1, 0)$	$\triangleright \mathcal{O}(1)$
7:	<b>si</b> $d = \text{DERECHA}$ <b>entonces</b>	$\triangleright \mathcal{O}(1)$
8:	$\text{desplazamiento} \leftarrow (1, 0)$	$\triangleright \mathcal{O}(1)$
9:	$\text{nuevaPosicion} \leftarrow p.\text{jugador} + \text{desplazamiento}$	$\triangleright \mathcal{O}(1)$
10:	<b>si</b> $\neg \text{mapa.esPared}(p.\text{mapa}, \text{nuevaPosicion}) \wedge \text{mapa.enRango}(p.\text{mapa}, \text{nuevaPosicion})$ <b>entonces</b>	$\triangleright \mathcal{O}(1)$
11:	$p.\text{jugador} \leftarrow \text{nuevaPosicion}$	$\triangleright \mathcal{O}(1)$
12:	$p.\text{cantMov} + +$	$\triangleright \mathcal{O}(1)$
13:	$\text{idChocolate} \leftarrow \text{mapa.idChocolate}(p.\text{mapa}, p.\text{jugador})$	$\triangleright \mathcal{O}(1)$
14:	$\text{hayUnChocolate} \leftarrow 0 \leq \text{idChocolate} < \text{mapa.cantChocolates}(p.\text{mapa})$	$\triangleright \mathcal{O}(1)$
15:	<b>si</b> $\text{hayUnChocolate} \wedge_L p.\text{chocolates}[\text{idChocolate}]$ <b>entonces</b>	$\triangleright \mathcal{O}(1)$
16:	$p.\text{chocolates}[\text{idChocolate}] \leftarrow \text{false}$	$\triangleright \mathcal{O}(1)$
17:	$p.\text{inmunidad} \leftarrow 10$	$\triangleright \mathcal{O}(1)$
18:	<b>else</b>	
19:	$p.\text{inmunidad} \leftarrow \max(p.\text{inmunidad} - 1, 0)$	$\triangleright \mathcal{O}(1)$
20:	$\text{partida.perdio} \leftarrow \text{mapa.esCasilleroPeligroso}(m, \text{partida.jugador}) \wedge \text{partida.inmunidad} = 0$	$\triangleright \mathcal{O}(1)$
21:	$\text{partida.gano} \leftarrow \text{mapa.llegada}(m) = \text{partida.jugador}$	$\triangleright \mathcal{O}(1)$

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

Justificación: No se recorre ninguna array, solo se accede a indices especificos. Los datos que provee el mapa, se encuentran guardados en arrays, que se complejidad de acceso es  $\mathcal{O}(1)$ , igualmente el acceso y modificación del array  $\text{partida.chocolates}$  es  $\mathcal{O}(1)$

---



---



---

**iGanó?**(in  $p$ : partida)  $\rightarrow res : bool$

1:	$res \leftarrow p.\text{gano}$	$\triangleright \Theta(1)$
----	--------------------------------	----------------------------

Complejidad:  $\Theta(1)$

---



---



---

**iPerdió?**(in  $p$ : partida)  $\rightarrow res : bool$

1:	$res \leftarrow p.\text{perdio}$	$\triangleright \Theta(1)$
----	----------------------------------	----------------------------

Complejidad:  $\Theta(1)$

---



---



---

**iJugador**(in  $p$ : partida)  $\rightarrow res : coordenada$

1:	$res \leftarrow p.\text{jugador}$	$\triangleright \Theta(1)$
----	-----------------------------------	----------------------------

Complejidad:  $\Theta(1)$

---



---



---

**iCantMov**(in  $p$ : partida)  $\rightarrow res : nat$

1:	$res \leftarrow p.\text{cantMov}$	$\triangleright \Theta(1)$
----	-----------------------------------	----------------------------

Complejidad:  $\Theta(1)$

---

## Servicios usados

- Modulo Mapa

# Modulo Fichín

## Interfaz

se explica con: FICHÍN

géneros: fichin.

### Operaciones básicas de fichin

NUEVOFICHIN(**in**  $m$ : mapa)  $\rightarrow res$  : fichin

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

**Post**  $\equiv \{res = nuevoFichin(m)\}$

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

**Descripción:** Genera un fichín

**Aliasing:** Recibe el mapa por referencia

NUEVAPARTIDA(**in/out**  $f$ : fichin, **in**  $j$ : jugador)  $\rightarrow res$  : bool

**Pre**  $\equiv \{f_0 = f\}$

**Post**  $\equiv \{res = \neg alguienJugando?(f) \wedge_L res \Rightarrow_L f = nuevaPartida(f_0, j)\}$

**Complejidad:**  $O(c)$

**Descripción:** Inicia una nueva partida

MOVER(**in/out**  $f$ : fichin, **in**  $d$ : dirección)  $\rightarrow res$  : bool

**Pre**  $\equiv \{f_0 = f\}$

**Post**  $\equiv \{res = alguienJugando?(f) \wedge_L res \Rightarrow_L f = mover(f_0, d)\}$

**Complejidad:**  $O(|J|)$  donde  $|J|$  es el más largo de los nombres de los jugadores

**Descripción:** Mueve en la dirección indicada

VERRANKING(**in**  $f$ : fichin)  $\rightarrow res$  : ranking

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

**Post**  $\equiv \{res = ranking(f)\}$

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

**Descripción:** Devuelve el ranking del fichín

**Aliasing:** Devuelve el ranking por referencia

OBJETIVO(**in**  $f$ : fichin)  $\rightarrow res$  : tupla<jugador, nat>

**Pre**  $\equiv \{alguienJugando?(f) \wedge def?(jugadorActual(f), ranking(f))\}$

**Post**  $\equiv \{res = objetivo(f)\}$

**Complejidad:**  $O(J \cdot |J|)$  donde  $J$  es la cantidad de jugadores y  $|J|$  es el más largo de los nombres de los jugadores

**Descripción:** Devuelve una tupla con el oponente y su puntaje

## Implementación

### Representación

fichín se representa con  $fch$

donde  $fch$  es  $\text{tupla}(\text{mapa}: \text{mp},$   
     $\text{alguienJugando}: \text{bool},$   
     $\text{jugadorActual}: \text{string},$   
     $\text{partidaActual}: \text{pt},$   
     $\text{ranking}: \text{dicc}(\text{string}, \text{nat}))$

### Invariante de representación

$\text{Rep} : fch \rightarrow \text{boolean}$

$\text{Rep}(e) \equiv \text{True} \iff$   
     $(e.\text{alguienJugando} \iff (\text{longitud}(e.\text{jugadorActual}) > 0 \wedge \neg e.\text{partida.gano} \wedge \neg e.\text{partida.perdio})) \wedge$   
     $(e.\text{pt.gano} \Rightarrow$   
         $\text{def?}(e.\text{jugadorActual}, e.\text{ranking}) \wedge_L$   
         $\text{obtener}(e.\text{jugadorActual}, e.\text{ranking}) \leq e.\text{partida.cantMov}) \wedge$   
     $(e.\text{mapa} = e.\text{pt.mapa})$

### Función de abstracción

$\text{Abs} : fch \ e \rightarrow \text{partida} \qquad \{\text{Rep}(e)\}$

$(\forall e : fch) \text{Abs}(e) =_{\text{obs}} f : \text{fichin} \mid \text{mapa}(f) = e.\text{mapa} \wedge$   
     $\text{alguienJugando}(f) = e.\text{alguienJugando} \wedge$   
     $\text{ranking}(f) = e.\text{ranking} \wedge$   
     $e.\text{alguienJugando} \Rightarrow_L \text{partidaActual}(f) = e.\text{partidaActual} \wedge$   
     $e.\text{alguienJugando} \Rightarrow_L \text{jugadorActual}(f) = e.\text{jugadorActual}$

## Algoritmos

---

---

**iNuevoFichin**(in  $m : \text{mapa}$ )  $\rightarrow res : \text{fichin}$

1:  $res \leftarrow \langle m, \text{False}, \text{NULL}, \text{NULL}, \text{vacío} \rangle \qquad \triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---

---

---

**iNuevaPartida**(in/out  $f : \text{fichin}$ , in  $a : \text{jugador}$ )  $\rightarrow res : \text{bool}$

$res \leftarrow \text{False}$

**si**  $\neg f.\text{alguienJugando}$  **entonces**

$f.\text{jugadorActual} \leftarrow a \qquad \triangleright \Theta(1)$

$f.\text{alguienJugando} \leftarrow \text{True} \qquad \triangleright \Theta(1)$

$f.\text{partidaActual} \leftarrow \text{partida.iNuevaPartida}(f.\text{mapa}) \qquad \triangleright \mathcal{O}(c)$

$res \leftarrow \text{True}$

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

Justificación: La complejidad de este algoritmo es igual a la complejidad de *iNuevaPartida*, que es  $\mathcal{O}(c)$ , con  $c = \#chocolates$ .

---

---



---

**iMover**(in/out  $f$ : fichin, in  $d$ : direccion)  $\rightarrow res$ : bool

```

1: si  $f.alguienJugando$  entonces
2:   partida.iMover(f.partidaActual, d)  $\triangleright \Theta(1)$ 
3:    $res \leftarrow True$ 
4:   si partida.iGanó?(f.partidaActual) entonces  $\triangleright \mathcal{O}(1)$ 
5:     si def?(f.jugadorActual, f.ranking) entonces  $\triangleright \mathcal{O}(|J|)$ 
6:       si obtener(f.jugadorActual, f.ranking) > partida.iCantMov(f.partidaActual) entonces  $\triangleright \mathcal{O}(|J|)$ 
7:         definir(f.jugadorActual, partida.iCantMov(f.partidaActual), f.ranking)  $\triangleright \mathcal{O}(|J|)$ 
8:       else definir(f.jugadorActual, partida.iCantMov(f.partidaActual), f.ranking)  $\triangleright \mathcal{O}(|J|)$ 
9:   else
10:     $res \leftarrow False$   $\triangleright \Theta(1)$ 

```

Complejidad:  $\mathcal{O}(|J|)$  si se realiza un movimiento que finaliza la partida

Justificación: El diccionario se implementa con un trie, que tiene  $\mathcal{O}(|J|)$  como complejidad de def?, acceder a significado y definir.

---



---



---

**iVerRanking**(in  $f$ : fichin)  $\rightarrow res$ : dicc

```

1:  $res \leftarrow f.ranking$   $\triangleright \Theta(1)$ 

```

Complejidad:  $\Theta(1)$

---



---



---

**iObjetivo**(in  $f$ : fichin, in  $j$ : jugador)  $\rightarrow res$ :  $\langle jugador, nat \rangle$

```

1:  $i \leftarrow 0$   $\triangleright \Theta(1)$ 
2:  $C \leftarrow obtenerClavesTrie(f.ranking)$   $\triangleright \Theta(J)$ 
3:  $S \leftarrow \emptyset$   $\triangleright \Theta(1)$ 
4: mientras  $i < long(C)$  hacer
5:   si obtener( $C[i]$ ,  $f.ranking$ ) < obtener( $jugador$ ,  $f.ranking$ ) entonces  $\triangleright \mathcal{O}(|J|)$ 
6:      $Ag(C[i], S)$   $\triangleright \Theta(1)$ 
7:      $i \leftarrow i + 1$   $\triangleright \Theta(1)$ 
8: si  $\emptyset?(S)$  entonces
9:    $res \leftarrow \langle jugador, obtener(f.ranking, jugador) \rangle$   $\triangleright \Theta(1)$ 
10: else
11:    $max \leftarrow obtener(f.ranking, S[0])$   $\triangleright \mathcal{O}(|J|)$ 
12:    $maxJugador \leftarrow S[0]$   $\triangleright \Theta(1)$ 
13:    $j \leftarrow 0$ 
14:   mientras  $j < long(S)$  hacer
15:     si obtener( $f.ranking$ ,  $S[j]$ )  $\geq max$  entonces  $\triangleright \mathcal{O}(|J|)$ 
16:        $max \leftarrow obtener(f.ranking, S[j])$   $\triangleright \mathcal{O}(|J|)$ 
17:        $maxJugador \leftarrow S[j]$   $\triangleright \Theta(1)$ 
18:    $j \leftarrow j + 1$ 
19:    $res \leftarrow \langle maxJugador, max \rangle$ 

```

Complejidad:  $\sum_{k \leq J} \mathcal{O}(|J|) + \mathcal{O}(|J|) + \sum_{k \in S} \mathcal{O}(|J|) = \mathcal{O}(|J| * J)$

Justificación: Los conjuntos  $C, S$  son implementados con una lista enlazada para que insertar elementos sea  $\mathcal{O}(1)$ .

---

## Servicios usados

- Modulo Mapa
- Modulo Partida
- Modulo Diccionario Trie

## Módulo Diccionario Trie( $\kappa, \sigma$ )

El módulo Diccionario Trie provee un diccionario básico en el que se puede definir, borrar, y testear si una clave está definida en tiempo  $\mathcal{O}(|J|)$ , donde  $J$  es la cantidad de elementos del diccionario y  $|J| = \max_{n \in J} |n|$ .

Para describir la complejidad de las operaciones, vamos a llamar  $copy(k)$  al costo de copiar el elemento  $k \in \kappa \cup \sigma$  y  $equal(k_1, k_2)$  al costo de evaluar si dos elementos  $k_1, k_2 \in \kappa$  son iguales (i.e.,  $copy$  y  $equal$  son funciones de  $\kappa \cup \sigma$  y  $\kappa \times \kappa$  en  $\mathbb{N}$ , respectivamente).<sup>1</sup>

## Interfaz

### parámetros formales

**géneros**  $\kappa, \sigma$

**función**  $\bullet = \bullet(\text{in } k_1 : \kappa, \text{in } k_2 : \kappa) \rightarrow res : \text{bool}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} (k_1 = k_2)\}$   
**Complejidad:**  $\Theta(equal(k_1, k_2))$   
**Descripción:** función de igualdad de  $\kappa$ 's

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

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

**se explica con:**  $\text{DICCIONARIO}(\kappa, \sigma)$ ,  $\text{ITERADOR BIDIRECCIONAL}(\text{TUPLA}(\kappa, \sigma))$ .

**géneros:**  $\text{dicc}(\kappa, \sigma)$ ,  $\text{itDicc}(\kappa, \sigma)$ .

### Operaciones básicas de diccionario

$\text{VACÍO}() \rightarrow res : \text{dicc}(\kappa, \sigma)$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{vacío}\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** genera un diccionario vacío.

$\text{DEFINIR}(\text{in/out } d : \text{dicc}(\kappa, \sigma), \text{in } k : \kappa, \text{in } s : \sigma)$   
**Pre**  $\equiv \{d =_{\text{obs}} d_0\}$   
**Post**  $\equiv \{d =_{\text{obs}} \text{definir}(d, k, s)\}$   
**Complejidad:**  $\mathcal{O}(|J|)$ , donde  $|J|$  es el nombre más largo entre los jugadores  
**Descripción:** define la clave  $k$  con el significado  $s$  en el diccionario.

$\text{DEFINIDO?}(\text{in } d : \text{dicc}(\kappa, \sigma), \text{in } k : \kappa) \rightarrow res : \text{bool}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{def?}(d, k)\}$   
**Complejidad:**  $\mathcal{O}(|J|)$ , donde  $|J|$  es el nombre más largo entre los jugadores  
**Descripción:** devuelve **true** si y sólo  $k$  está definido en el diccionario.

$\text{SIGNIFICADO}(\text{in } d : \text{dicc}(\kappa, \sigma), \text{in } k : \kappa) \rightarrow res : \sigma$   
**Pre**  $\equiv \{\text{def?}(d, k)\}$   
**Post**  $\equiv \{\text{alias}(res =_{\text{obs}} \text{Significado}(d, k))\}$   
**Complejidad:**  $\mathcal{O}(|J|)$ , donde  $|J|$  es el nombre más largo entre los jugadores  
**Descripción:** devuelve el significado de una clave en el diccionario.

$\text{BORRAR}(\text{in/out } d : \text{dicc}(\kappa, \sigma), \text{in } k : \kappa)$   
**Pre**  $\equiv \{d = d_0 \wedge \text{def?}(d, k)\}$   
**Post**  $\equiv \{d =_{\text{obs}} \text{borrar}(d_0, k)\}$   
**Complejidad:**  $\mathcal{O}(|J|)$ , donde  $|J|$  es el nombre más largo entre los jugadores  
**Descripción:** elimina la clave  $k$  y su significado de  $d$ .

<sup>1</sup>Nótese que este es un abuso de notación, ya que no estamos describiendo  $copy$  y  $equal$  en función del tamaño de  $k$ . A la hora de usarlo, habrá que realizar la traducción.

**OBTENERCLAVESDELTRIE**(**in**  $d : \text{dicc}(\kappa, \sigma) \rightarrow res : \text{conj}(\kappa)$ )

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

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

**Complejidad:**  $\Theta(J)$

**Descripción:** devuelve el conjunto de claves del diccionario.

**#CLAVES**(**in**  $d : \text{dicc}(\kappa, \sigma) \rightarrow res : \text{nat}$ )

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

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

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve la cantidad de claves del diccionario.

**COPIAR**(**in**  $d : \text{dicc}(\kappa, \sigma) \rightarrow res : \text{dicc}(\kappa, \sigma)$ )

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

**Post**  $\equiv \{res =_{\text{obs}} d\}$

**Complejidad:**  $\Theta \left( \sum_{k \in K} (\text{copy}(k) + \text{copy}(\text{significado}(k, d))) \right)$ , donde  $K = \text{claves}(d)$

**Descripción:** genera una copia nueva del diccionario.

**• = •**(**in**  $d_1 : \text{dicc}(\kappa, \sigma)$ , **in**  $d_2 : \text{dicc}(\kappa, \sigma) \rightarrow res : \text{bool}$ )

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

**Post**  $\equiv \{res =_{\text{obs}} c_1 = c_2\}$

**Complejidad:**  $O \left( \sum_{\substack{k_1 \in K_1 \\ k_2 \in K_2}} \text{equal}(\langle k_1, s_1 \rangle, \langle k_2, s_2 \rangle) \right)$ , donde  $K_i = \text{claves}(d_i)$  y  $s_i = \text{significado}(d_i, k_i)$ ,  $i \in \{1, 2\}$ .

**Descripción:** compara  $d_1$  y  $d_2$  por igualdad, cuando  $\sigma$  posee operación de igualdad.

**Requiere:** **• = •**(**in**  $s_1 : \sigma$ , **in**  $s_2 : \sigma \rightarrow res : \text{bool}$ )

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

**Post**  $\equiv \{res =_{\text{obs}} (s_1 = s_2)\}$

**Complejidad:**  $\Theta(\text{equal}(s_1, s_2))$

**Descripción:** función de igualdad de  $\sigma$ 's