

# Algoritmos y Estructuras de Datos II

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

## Trabajo Práctico 3

### Grupo Omega

Integrante	LU	Correo electrónico
Candia, Matias	721/19	candia.matias2000@gmail.com
Caneva, Diego Gabriel	169/18	diego.g.caneva@gmail.com
Lin Zabala, Juan Ignacio	349/18	juanignacio.lin@gmail.com
Sarmiento, Matias Federico	741/18	matiasfsarmiento@gmail.com

### Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

# 1. Módulo Mapa

## Interfaz

se explica con: MAPA.

géneros: mapa.

Operaciones:

**CREARMAPA**(**in** *alto* : nat, **in** *ancho* : nat, **in** *s* : coordenada, **in** *ll* : coordenada, **in** *f* : conj (coordenada),  
**in** *p* : conj (coordenada), **in** *ch* : conj (coordenada))  $\rightarrow$  *res* : mapa  
**Pre**  $\equiv \{s \neq ll \wedge \text{todosEnRango}(p \cup f \cup ch \cup \{s, ll\}, ancho, alto) \wedge \{s, ll\} \cap (f \cup p) = \emptyset \wedge \text{disjuntosDeAPares}(p, f, ch)\}$   
**Post**  $\equiv \{res = \text{nuevoMapa}(ancho, alto, s, ll, p, f, ch)\}$   
**Complejidad**:  $\Theta(alto \times ancho)$   
**Descripción**: Crea una nueva instancia de mapa

**RESETEARCHOCOS**(**in/out** *m* : mapa)  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{\text{Restaura los chocolates a como fue creado originalmente}\}$   
**Complejidad**:  $\Theta(c)$   
**Descripción**: Pone en el mapa chocolates en las coordenadas del conjunto chocosOri

**SALIDA**(**in** *m* : mapa)  $\rightarrow$  *res* : coordenada  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res = \text{inicio}(m)\}$   
**Complejidad**:  $\Theta(1)$   
**Descripción**: Devuelve la salida del mapa

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

**FANTASMAS**(**in** *m* : mapa)  $\rightarrow$  *res* : arreglo(arreglo(bool))  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{(\forall c : \text{coordenada})(\text{enRango}(c, largo(m), alto(m)) \Rightarrow_L (c \in \text{fantasmas}(m) \iff res[c] = \text{true}))\}$   
**Complejidad**:  $\Theta(1)$   
**Descripción**: Devuelve una matriz de bool con true donde hay fantasmas y false donde no  
**Aliasing**: Devuelve una referencia no modificable

**PAREDES**(**in** *m* : mapa)  $\rightarrow$  *res* : arreglo(arreglo(bool))  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{(\forall c : \text{coordenada})(\text{enRango}(c, largo(m), alto(m)) \Rightarrow_L (c \in \text{paredes}(m) \iff res[c] = \text{true}))\}$   
**Complejidad**:  $\Theta(1)$   
**Descripción**: Devuelve una matriz de bool con true donde hay paredes y false donde no  
**Aliasing**: Devuelve una referencia no modificable

**CHOCOLATES**(**in**  $m : \text{mapa}$ )  $\rightarrow res : \text{arreglo}(\text{arreglo}(\text{bool}))$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{(\forall c : \text{coordenada})(\text{enRango}(c, \text{largo}(m), \text{alto}(m)) \Rightarrow_{\text{L}} (res[c] = \text{true} \Rightarrow c \in \text{chocolates}(m)))\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** Devuelve una matriz de bool con true donde hay chocolates y false donde no  
**Aliasing:** Devuelve una referencia modificable

## Representación

**mapa se representa con estrMapa**

donde **estrMapa** es  $\text{tupla}(\text{alto: nat} , \text{ancho: nat} , \text{salida: coordenada} , \text{llegada: coordenada} , \text{chocolates: arreglo}(\text{arreglo}(\text{bool})) , \text{fantasmas: arreglo}(\text{arreglo}(\text{bool})) , \text{paredes: arreglo}(\text{arreglo}(\text{bool})) , \text{chocosOri: conj}(\text{coordenada}) )$

**Rep** :  $\text{estrMapa} \rightarrow \text{bool}$

**Rep**( $m$ )  $\equiv \text{true} \iff$

1.  $m.\text{chocolates}$ ,  $m.\text{fantasmas}$  y  $m.\text{paredes}$  tienen dimensión  $m.\text{alto} \times m.\text{ancho}$ .
2.  $m.\text{salida}$  y  $m.\text{llegada}$  son coordenadas en rango.
3.  $m.\text{fantasmas}$  y  $m.\text{paredes}$  tienen falso al indexarlos en  $m.\text{salida}$  y en  $m.\text{llegada}$ .
4. Las coordenadas con valor true de  $m.\text{chocolates}$  son elementos de  $m.\text{chocosOri}$ .
5. No hay ninguna coordenada con valor true en más de una de las matrices  $m.\text{chocolates}$ ,  $m.\text{fantasmas}$  y  $m.\text{paredes}$ .
6. No hay ninguna coordenada de  $m.\text{chocosOri}$  con valor true en las matrices  $m.\text{fantasmas}$  y  $m.\text{paredes}$ .

**Abs** :  $\text{estrMapa } e \rightarrow \text{mapa}$

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

$$\text{Abs}(e) \equiv m: \text{mapa} \mid \text{largo}(m) = e.\text{ancho} \wedge \text{alto}(m) = e.\text{alto} \wedge \text{inicio}(m) = e.\text{salida} \wedge \text{llegada}(m) = e.\text{llegada} \wedge (\forall c: \text{coordenada})(c \in \text{paredes}(m) \iff e.\text{paredes}[c] = \text{true}) \wedge (\forall c: \text{coordenada})(c \in \text{fantasmas}(m) \iff e.\text{fantasmas}[c] = \text{true}) \wedge \text{chocolates}(m) = e.\text{chocosOri}$$

## Algoritmos

---

```

crearMapa(in alto: nat, in ancho: nat, in s: coordenada, in ll: coordenada, in f: conj(coordenada),
in p: conj(coordenada), in ch: conj(coordenada)) → res: mp
1: res.alto ← alto                                ▷ Θ(1)
2: res.ancho ← ancho                                ▷ Θ(1)
3: res.salida ← s                                    ▷ Θ(1)
4: res.llegada ← ll                                  ▷ Θ(1)
5: res.chocosOri ← ch                                ▷ Θ(c)
6: res.fantasmas ← inicializar(alto, inicializar(ancho, false))    ▷ Θ(ancho × alto)
7: res.paredes ← inicializar(alto, inicializar(ancho, false))    ▷ Θ(ancho × alto)
8: res.chocolates ← inicializar(alto, inicializar(ancho, false))    ▷ Θ(ancho × alto)
9: itConj(coordenada) itF ← crearIt(f)                ▷ Θ(1)
10: while haySiguiente(itF) do                        ▷ Θ(1). El ciclo se hace #f veces
11:   x ← siguiente(itF).first, y ← siguiente(itF).second;    ▷ Θ(1)
12:   for j ← max(−3, −x); j ≤ min(3, res.alto − 1 − x); ++ j do
13:     for k ← max(−3, −y); k ≤ min(3, res.ancho − 1 − y); ++ k do
14:       if abs(k) + abs(j) ≤ 3 then
15:         res.fantasmas[x + j][y + k] ← true
16:       end if
17:     end for
18:   end for
19:   avanzar(itF)                                ▷ Θ(1)
20: end while
21: itConj(coordenada) itP ← crearIt(p)                ▷ Θ(1)
22: while haySiguiente(itP) do                        ▷ Θ(1). El ciclo se hace #p veces
23:   res.paredes[siguiente(itP)] ← true                ▷ Θ(1)
24:   avanzar(itP)                                    ▷ Θ(1)
25: end while
26: itConj(coordenada) itCh ← crearIt(ch)                ▷ Θ(1)
27: while haySiguiente(itCh) do                        ▷ Θ(1). El ciclo se hace c veces
28:   res.chocolates[siguiente(itCh)] ← true            ▷ Θ(1)
29:   avanzar(itCh)                                    ▷ Θ(1)
30: end while

```

Complejidad: Θ(*alto* × *ancho*), ya que #*f*, #*p* y *c* ≤ *alto* × *ancho*

---

---



---

**resetearChocos**(in/out  $m : \text{mp}$ )

1: $itConj(coordenada) \text{ } itCh \leftarrow crearIt(m.chocosOri)$	$\triangleright \Theta(1)$
2: <b>while</b> $haySiguiente(itCh)$ <b>do</b>	$\triangleright \Theta(c)$
3: $res.chocolates[siguiente(itCh)] \leftarrow true$	$\triangleright \Theta(1)$
4: $avanzar(itCh)$	$\triangleright \Theta(1)$
5: <b>end while</b>	

Complejidad:  $\Theta(c)$

---



---



---

**salida**(in  $m : \text{mp}$ )  $\rightarrow res : \text{coordenada}$

1: $res \leftarrow m.salida$	$\triangleright \Theta(1)$
------------------------------	----------------------------

Complejidad:  $\Theta(1)$

---



---



---

**llegada**(in  $m : \text{mp}$ )  $\rightarrow res : \text{coordenada}$

1: $res \leftarrow m.llegada$	$\triangleright \Theta(1)$
-------------------------------	----------------------------

Complejidad:  $\Theta(1)$

---



---



---

**fantasmas**(in  $m : \text{mp}$ )  $\rightarrow res : \text{arreglo}(\text{arreglo}(\text{bool}))$

1: $res \leftarrow m.fantasmas$	$\triangleright \Theta(1)$
---------------------------------	----------------------------

Complejidad:  $\Theta(1)$   
Es  $\Theta(1)$  porque retorna un referencia (no modificable)

---



---



---

**paredes**(in  $m : \text{mp}$ )  $\rightarrow res : \text{arreglo}(\text{arreglo}(\text{bool}))$

1: $res \leftarrow m.paredes$	$\triangleright \Theta(1)$
-------------------------------	----------------------------

Complejidad:  $\Theta(1)$   
Es  $\Theta(1)$  porque retorna un referencia (no modificable)

---



---



---

**chocolates**(in  $m : \text{mp}$ )  $\rightarrow res : \text{arreglo}(\text{arreglo}(\text{bool}))$

1: $res \leftarrow m.chocolates$	$\triangleright \Theta(1)$
----------------------------------	----------------------------

Complejidad:  $\Theta(1)$   
Es  $\Theta(1)$  porque retorna un referencia (modificable)

---

## Servicios Usados

Del módulo Conjunto Lineal( $\alpha$ ) se utilizó:

- **CrearIt**(in c: conj( $\alpha$ )): crea un iterador bidireccional del conjunto, de forma tal que HayAnterior evalúe a false. Tiene una complejidad  $\Theta(1)$ .
- **haySiguiente**(in it: itConj( $\alpha$ )): devuelve true si y sólo si en el iterador todavía quedan elementos para avanzar. Tiene una complejidad  $\Theta(1)$ .
- **Siguiente**(in it: itConj( $\alpha$ )): devuelve el elemento siguiente a la posición del iterador. Tiene una complejidad  $\Theta(1)$ .
- **Avanzar**( in/out it: itConj( $\alpha$ )): avanza el iterador a la posición siguiente. Tiene una complejidad  $\Theta(1)$ .

Del módulo Arreglo( $\alpha$ ) se utilizó:

- **•[•]**: devuelve o define el elemento que se encuentra en la  $i$ -ésima posición del arreglo en base 0. Tiene una complejidad de  $\Theta(1)$ .
- Extendemos el módulo Arreglo con la siguiente operación:

**INICIALIZAR**(in  $dim : \text{nat}$ , in  $a : \alpha$ )  $\rightarrow res : \text{arreglo}(\alpha)$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{long(res) = dim \wedge (\forall i : \text{nat})(0 \leq i < dim \Rightarrow_L res[i] == a)\}$   
**Complejidad**:  $\Theta(dim * copy(a))$   
**Descripción**: Devuelve un arreglo de  $dim$  posiciones todas con valor  $a$   
**Aliasing**: No aplica

## 2. Módulo Partida

### Interfaz

**se explica con**: PARTIDA.

**géneros**: partida.

Operaciones:

**CREARPARTIDA**(in  $m : \text{mapa}$ )  $\rightarrow res : \text{partida}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res = nuevaPartida(m)\}$   
**Complejidad**:  $\Theta(1)$   
**Descripción**: Crea una nueva instancia de partida  
**Aliasing**: Mapa es pasado por referencia modificable

**RESETTEARPARTIDA**(in/out  $p : \text{partida}$ )  
**Pre**  $\equiv \{p = P_0\}$   
**Post**  $\equiv \{p = nuevaPartida(resetearChocos(mapa(P_0)))\}$   
**Complejidad**:  $\Theta(c)$   
**Descripción**: Resetear los chocolates en el mapa de la partida  
**Aliasing**: Modifica el mapa de la partida

**MOVESE**(**in/out**  $p$ : partida, **in**  $d$ : dir)  
**Pre**  $\equiv \{p = P_0 \wedge \neg(gano?(p) \vee perdio?(p))\}$   
**Post**  $\equiv \{p = mover(P_0, d)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** Mueve el jugador en la dirección  $d$ , actualiza la inmunidad según corresponda, si come un chocolate lo borra del mapa, actualiza el puntaje  
**Aliasing:** Modifica el mapa de partida en caso de comer un chocolate

**ACTUAL**(**in**  $p$ : partida)  $\rightarrow res$  : coordenada  
**Pre**  $\equiv \{true\}$   
**Post**  $\equiv \{res = jugador(p)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** Retorna la coordenada actual del jugador  
**Aliasing:** No aplica

**GANO?**(**in**  $p$ : partida)  $\rightarrow res$  : bool  
**Pre**  $\equiv \{true\}$   
**Post**  $\equiv \{res = gano?(p)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** Retorna si el jugador ganó la partida  
**Aliasing:** No aplica

**PERDIO?**(**in**  $p$ : partida)  $\rightarrow res$  : bool  
**Pre**  $\equiv \{true\}$   
**Post**  $\equiv \{res = perdio?(p)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** Retorna si el jugador perdió la partida  
**Aliasing:** No aplica

**PUNTAJE**(**in**  $p$ : partida)  $\rightarrow res$  : nat  
**Pre**  $\equiv \{true\}$   
**Post**  $\equiv \{res = cantMov(p)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** Retorna el puntaje de la partida  
**Aliasing:** No aplica

## Representación

partida se representa con **estrPartida**

donde **estrPartida** es  $tupla(mapa: mapa, actual: coordenada, inmunidad: nat, puntaje: nat)$

**Rep** : **estrPartida**  $\rightarrow$  bool

**Rep**( $p$ )  $\equiv true \iff$

1.  $0 \leq inmunidad \leq 10$ .
2. *actual* debe ser una coordenada válida.
3. *puntaje* debe ser mayor o igual que  $distManhattan(salida, actual)$ .

**Abs** : **estrPartida**  $e \rightarrow$  partida

$\{Rep(e)\}$

$$\text{Abs}(e) \equiv p: \text{partida} \mid \text{mapa}(p) = e.\text{mapa} \wedge \text{jugador}(p) = e.\text{actual} \wedge \text{cantMov}(p) = e.\text{puntaje} \wedge \text{inmunidad}(p) = e.\text{inmunidad} \wedge_L (\forall c : \text{coordenada})(c \in \text{chocolates}(p) \iff \text{chocolates}(e.\text{mapa})[c] == \text{true})$$

## Algoritmos

---



---

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

1: $res.\text{mapa} \leftarrow m$	▷ $\Theta(1)$
2: $res.\text{actual} \leftarrow \text{salida}(m)$	▷ $\Theta(1)$
3: $res.\text{inmunidad} \leftarrow 0$	▷ $\Theta(1)$
4: <b>if</b> $\text{chocolates}(m)[res.\text{actual}]$ <b>then</b>	▷ $\Theta(1)$
5: $res.\text{inmunidad} \leftarrow 10$	▷ $\Theta(1)$
6: $\text{chocolates}(m)[res.\text{actual}] \leftarrow \text{false}$	▷ $\Theta(1)$
7: <b>end if</b>	
8: $res.\text{puntaje} \leftarrow 0$	▷ $\Theta(1)$

Complejidad:  $\Theta(1)$

$res.\text{mapa}$  se le asigna  $m$  por referencia

---



---



---

**resetearPartida**(in/out  $p: \text{partida}$ )

1: $\text{resetearChocos}(res.\text{mapa}) \leftarrow m$	▷ $\Theta(c)$
2: $res.\text{actual} \leftarrow \text{salida}(m)$	▷ $\Theta(1)$
3: $res.\text{inmunidad} \leftarrow 0$	▷ $\Theta(1)$
4: <b>if</b> $\text{chocolates}(m)[res.\text{actual}]$ <b>then</b>	▷ $\Theta(1)$
5: $res.\text{inmunidad} \leftarrow 10$	▷ $\Theta(1)$
6: $\text{chocolates}(m)[res.\text{actual}] \leftarrow \text{false}$	▷ $\Theta(1)$
7: <b>end if</b>	
8: $res.\text{puntaje} \leftarrow 0$	▷ $\Theta(1)$

Complejidad:  $\Theta(c)$

---



---



---

```

move(in/out  $p$ : partida, in  $d$ : dir)
1:  $\text{coordenada destino} \leftarrow p.\text{actual}$   $\triangleright \Theta(1)$ 
2: if  $d == \text{DERECHA}$  then  $\triangleright \Theta(1)$ 
3:    $\text{destino} = \text{destino} + \langle 0, 1 \rangle$   $\triangleright \Theta(1)$ 
4: end if
5: if  $d == \text{IQUIERDA}$  then  $\triangleright \Theta(1)$ 
6:    $\text{destino} = \text{destino} + \langle 0, -1 \rangle$   $\triangleright \Theta(1)$ 
7: end if
8: if  $d == \text{ABAJO}$  then  $\triangleright \Theta(1)$ 
9:    $\text{destino} = \text{destino} + \langle 1, 0 \rangle$   $\triangleright \Theta(1)$ 
10: end if
11: if  $d == \text{ARRIBA}$  then  $\triangleright \Theta(1)$ 
12:    $\text{destino} = \text{destino} + \langle -1, 0 \rangle$   $\triangleright \Theta(1)$ 
13: end if
14: if  $0 \leq \pi_0(\text{destino}) < \text{alto}(p.\text{mapa}) \wedge 0 \leq \pi_1(\text{destino}) < \text{ancho}(p.\text{mapa})$  then  $\triangleright \Theta(1)$ 
15:   if  $\neg \text{paredes}(p.\text{mapa})[\text{destino}]$  then  $\triangleright \Theta(1)$ 
16:      $p.\text{actual} \leftarrow \text{destino}$   $\triangleright \Theta(1)$ 
17:      $\text{res.puntaje} ++$   $\triangleright \Theta(1)$ 
18:     if  $\text{chocolates}(m)[p.\text{actual}]$  then  $\triangleright \Theta(1)$ 
19:        $p.\text{inmunidad} \leftarrow 10$   $\triangleright \Theta(1)$ 
20:        $\text{chocolates}(m)[p.\text{actual}] \leftarrow \text{false}$   $\triangleright \Theta(1)$ 
21:     else
22:       if  $\text{inmunidad} > 0$  then
23:          $p.\text{inmunidad} --$   $\triangleright \Theta(1)$ 
24:       end if
25:     end if
26:   end if
27: end if

Complejidad:  $\Theta(1)$ 

```

---



---



---

```

actual(in  $p$ : partida)  $\rightarrow \text{res}$  : coordenada
1:  $\text{res} \leftarrow p.\text{actual}$   $\triangleright \Theta(1)$ 

Complejidad:  $\Theta(1)$ 

```

---



---



---

```

gano?(in  $p$ : partida)  $\rightarrow \text{res}$  : bool
1:  $\text{res} \leftarrow p.\text{actual} == \text{llegada}(p.\text{mapa})$   $\triangleright \Theta(1)$ 

Complejidad:  $\Theta(1)$ 

```

---

---

---

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

1:  $res \leftarrow fantasma(p.mapa)[p.actual] \wedge p.inmunidad == 0$   $\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---

---

---

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

1:  $res \leftarrow p.puntaje$   $\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---

## Servicios Usados

Del módulo Arreglo( $\alpha$ ) se utilizó:

- $\bullet[\bullet]$ : devuelve el elemento que se encuentra en la  $i$ -ésima posición del arreglo en base 0. Tiene una complejidad de  $\Theta(1)$ .

### 3. Módulo Fichin

#### Interfaz

**se explica con:** FICHIN.

**géneros:** fichin.

Operaciones:

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

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

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

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

**Descripción:** Crea una nueva instancia de fichin

**Aliasing:** Mapa es pasado por referencia modificable

NUEVAPARTIDA(**in/out**  $f : \text{fichin}$ , **in**  $string : j$ )

**Pre**  $\equiv \{f = F_0 \wedge \neg(\text{alguienJugando?}(f))\}$

**Post**  $\equiv \{f = \text{nuevaPartida}(F_0, j)\}$

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

**Descripción:** Genera una nueva instancia de Partida dentro del fichin  $f$  con jugador  $j$

**Aliasing:** Resetea los chocolates del mapa

MOVER(**in/out**  $f : \text{fichin}$ , **in**  $dir : d$ )

**Pre**  $\equiv \{f = F_0 \wedge \text{alguienJugando?}(f)\}$

**Post**  $\equiv \{f = \text{mover}(F_0, d)\}$

**Complejidad:** Si se ganó la partida es  $\Theta(|J|)$ , caso contrario es  $\Theta(1)$

**Descripción:** Se mueve la posición del jugador dentro de la partida, siempre que la dirección  $d$  de un movimiento válido

**Aliasing:** En caso de comerse un chocolate se modifica el mapa

VERRANKING(**in**  $f : \text{fichin}$ )  $\rightarrow res : \text{lista}(\langle \text{string}, \text{nat} \rangle)$

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

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

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

**Descripción:** Retorna un diccionario con el nombre y el mejor puntaje de cada jugador que haya ganado alguna partida

**Aliasing:** Se retorna una referencia no modicable

OBJETIVO(**in**  $f : \text{fichin}$ )  $\rightarrow res : \langle \text{string}, \text{nat} \rangle$

**Pre**  $\equiv \{\text{alguienJugando?}(f) \wedge_{\text{L}} \text{definido}(\text{ranking}(f), \text{jugadorActual}(f))\}$

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

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

**Descripción:** Retorna el jugador cuyo puntaje en el ranking es inmediatamente mejor al puntaje en el ranking del jugador actual y su respectivo puntaje

**Aliasing:** No aplica

## Representación

**fichin se representa con estrFichin**

donde **estrFichin** es `tupla(partida: partida , jugando: bool , jugador: string , jugadores: diccTrie(string, nat) )`

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

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

1. Si *jugando* es true, entonces *jugador* no es vacío y *partida* no puede estar ganada ni perdida.

$\text{Abs} : \text{estrFichin } e \rightarrow \text{fichin}$

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

$\text{Abs}(e) \equiv f: \text{fichin} \mid \text{mapa}(f) = \text{reiniciarChocos}(e.\text{mapa}) \wedge \text{alguienJugando}(f) = e.\text{jugando} \wedge_{\text{L}} (\text{alguienJugando}(f) = \text{true} \Rightarrow_{\text{L}} \text{jugadorActual}(f) = e.\text{jugador}) \wedge \text{partidaActual}(f) = e.\text{partida} \wedge \text{claves}(\text{ranking}(f)) = \text{claves}(e.\text{jugadores}) \wedge_{\text{L}} (\forall j : \text{string})(j \in \text{claves}(\text{ranking}(f))) \Rightarrow_{\text{L}} \text{significado}(\text{ranking}(f), j) = \text{puntos}(\text{Siguiente}(\text{significado}(e.\text{jugadores}, j))))$

## Algoritmos

---

---

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

- |   |                            |
|---|----------------------------|
| 1: $res.\text{partida} \leftarrow \text{crearPartida}(m)$ | $\triangleright \Theta(1)$ |
| 2: $res.\text{jugando} \leftarrow \text{false}$           | $\triangleright \Theta(1)$ |
| 3: $res.\text{jugador} \leftarrow ""$                     | $\triangleright \Theta(1)$ |
| 4: $res.\text{jugadores} \leftarrow \text{vacío}()$       | $\triangleright \Theta(1)$ |
| 5: $res.\text{ranking} \leftarrow \text{vacío}()$         | $\triangleright \Theta(1)$ |

Complejidad:  $\Theta(1)$

A  $res.\text{mapa}$  se le asigna una referencia a  $m$

---

---

---

**nuevaPartida**(in/out  $f : \text{fichin}$ , in  $j : \text{string}$ )

- |   |                            |
|---|----------------------------|
| 1: $f.\text{partida} \leftarrow \text{resetearPartida}(f.\text{partida})$ | $\triangleright \Theta(c)$ |
| 2: $f.\text{jugando} \leftarrow \text{true}$                              | $\triangleright \Theta(1)$ |
| 3: $f.\text{jugador} \leftarrow j$  | $\triangleright \Theta(1)$ |
| 4: $f.\text{puesto} = \text{crearIt}(f.\text{ranking})$                   | $\triangleright \Theta(1)$ |

Complejidad:  $\Theta(c)$

---

---



---

```

mover(in/out  $f : \text{fichin}$ , in  $d : \text{dir}$ )
1:  $\text{move}(f.\text{partida}, d)$   $\triangleright \Theta(1)$ 
2: if  $\text{HaySiguiete}(f.\text{puesto}) \wedge_{\text{L}} \text{puntaje}(f.\text{partida}) \geq \text{puntos}(\text{Siguiete}(f.\text{puesto}))$  then  $\triangleright \Theta(1)$ 
3:    $\text{Avanzar}(f.\text{puesto})$   $\triangleright \Theta(1)$ 
4: end if
5: if  $\text{gano?}(f.\text{partida})$  then  $\triangleright \Theta(1)$ 
6:   if  $\text{definido?}(f.\text{jugadores}, f.\text{jugador})$  then  $\triangleright \Theta(|J|)$ 
7:     if  $\text{puntos}(\text{Siguiete}(\text{significado}(f.\text{jugadores}, f.\text{jugador}))) > \text{puntaje}(f.\text{partida})$  then  $\triangleright \Theta(|J|)$ 
8:        $\text{EliminarSiguiete}(\text{significado}(f.\text{jugadores}, f.\text{jugador}))$   $\triangleright \Theta(|J|)$ 
9:        $\text{AgregarComoSiguiete}(f.\text{puesto}, \langle f.\text{jugador}, \text{puntaje}(f.\text{partida}) \rangle)$   $\triangleright \Theta(1)$ 
10:       $\text{definir}(f.\text{jugadores}, f.\text{jugador}, f.\text{puesto})$   $\triangleright \Theta(|J|)$ 
11:    end if
12:  else
13:     $\text{AgregarComoSiguiete}(f.\text{puesto}, \langle f.\text{jugador}, \text{puntaje}(f.\text{partida}) \rangle)$   $\triangleright \Theta(1)$ 
14:     $\text{definir}(f.\text{jugadores}, f.\text{jugador}, f.\text{puesto})$   $\triangleright \Theta(|J|)$ 
15:  end if
16: end if
17:  $f.\text{jugando} \leftarrow \neg (\text{gano?}(f.\text{partida}) \vee \text{perdio?}(f.\text{partida}))$   $\triangleright \Theta(1)$ 

```

Complejidad:  $\Theta(|J|)$

---



---



---



---

```

verRanking(in  $f : \text{fichin}$ )  $\rightarrow res : \text{lista}(\langle \text{string}, \text{nat} \rangle)$ 
1:  $res \leftarrow f.\text{ranking}$   $\triangleright \Theta(1)$ 

```

Complejidad:  $\Theta(1)$   
A  $res$  se le asigna un referencia no modificable

---



---

---



---

<b>objetivo</b> (in $f: \text{fichin}$ ) $\rightarrow res: \langle \text{string}, \text{nat} \rangle$	
1: $puntos \leftarrow \text{significado}(f.\text{jugadores}, f.\text{jugador})$	$\triangleright \Theta( J )$
2: $obj2 \leftarrow 0$	$\triangleright \Theta(1)$
3: $itDiccTrie(string, nat) \text{ } itR \leftarrow \text{crearIt}(f.\text{jugadores})$	$\triangleright \Theta(1)$
4: <b>while</b> $\text{haySiguiente}(itR)$ <b>do</b>	$\triangleright \Theta(1)$ . El ciclo se hace $J$ veces
5: $sig \leftarrow \text{siguienteSignificado}(itR)$	$\triangleright \Theta(1)$
6: <b>if</b> $obj2 \leq sig \wedge sig < puntos$ <b>then</b>	$\triangleright \Theta(1)$
7: $obj2 \leftarrow sig$	$\triangleright \Theta(1)$
8: $obj1 \leftarrow \text{siguienteClave}(itR)$	$\triangleright \Theta(1)$
9: <b>end if</b>	
10: $\text{avanzar}(itR)$	$\triangleright \Theta(1)$
11: <b>end while</b>	
12: <b>if</b> $obj2 == 0$ <b>then</b>	$\triangleright \Theta(1)$
13: $obj2 \leftarrow puntos$	$\triangleright \Theta(1)$
14: $obj1 \leftarrow f.\text{jugador}$	$\triangleright \Theta(1)$
15: <b>end if</b>	
16: $res \leftarrow \langle obj1, obj2 \rangle$	$\triangleright \Theta(1)$
Complejidad: $\Theta(J)$	

---

## Servicios Usados

Del módulo Lista Enlazada( $\alpha$ ) se utilizó:

- $\text{CrearIt}(\text{in } l: \text{lista}(\alpha))$ : crea un iterador bidireccional de la lista, de forma tal que al pedir Siguiente se obtenga el primer elemento de  $l$ . Tiene una complejidad  $\Theta(1)$ .
- $\text{haySiguiente}(\text{in } it: \text{itLista}(\alpha))$ : devuelve true si y sólo si en el iterador todavía quedan elementos para avanzar. Tiene una complejidad  $\Theta(1)$ .
- $\text{Siguiente}(\text{in } it: \text{itLista}(\alpha))$ : devuelve el elemento siguiente a la posición del iterador. Tiene una complejidad  $\Theta(1)$ .
- $\text{hayAnterior}(\text{in } it: \text{itLista}(\alpha))$ : devuelve true si y sólo si en el iterador todavía quedan elementos para retroceder. Tiene una complejidad  $\Theta(1)$ .
- $\text{Anterior}(\text{in } it: \text{itLista}(\alpha))$ : devuelve el elemento anterior del iterador. Tiene una complejidad  $\Theta(1)$ .
- $\text{Avanzar}(\text{ in/out } it: \text{itLista}(\alpha))$ : avanza el iterador a la posición siguiente. Tiene una complejidad  $\Theta(1)$ .
- $\text{EliminarSiguiente}(\text{ in/out } it: \text{itLista}(\alpha))$ : elimina de la lista iterada el valor que se encuentra en la posición siguiente del iterador. Tiene una complejidad  $\Theta(1)$ .
- $\text{AgregarComoSiguiente}(\text{ in/out } it: \text{itLista}(\alpha), \text{ in } a: \alpha)$ : agrega el elemento  $a$  a la lista iterada, entre las posiciones anterior y siguiente del iterador, dejando al iterador posicionado de forma tal que al llamar a Siguiente se obtenga  $a$ . Tiene una complejidad  $(\Theta(\text{copy}(a)))$ .

El módulo Diccionario Trie ( $\kappa, \sigma$ ) es un diccionario implementado sobre un trie con la siguiente interfaz:

**se explica con:** DICCIONARIO.

**géneros:** diccTrie.

DEFINIR(**in/out**  $d: \text{dicc}(\kappa, \sigma)$ , **in**  $k: \kappa$ , **in**  $s: \sigma$ )

**Pre**  $\equiv \{d = D_0\}$

**Post**  $\equiv \{d = \text{definir}(D_0, k, s)\}$

**Complejidad:**  $\Theta(\text{longitud del elemento de tipo } \kappa \text{ más largo de las claves de } d) + \Theta(\text{copy}(s))$ .

**Descripción:** define la clave  $k$  con el significado  $s$  en el diccionario.

**Aliasing:** los elementos  $k$  y  $s$  se definen por copia.

DEFINIDO?(**in**  $d: \text{dicc}(\kappa, \sigma)$ , **in**  $k: \kappa$ )  $\rightarrow res: \text{bool}$

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

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

**Complejidad:**  $\Theta(\text{longitud del elemento de tipo } \kappa \text{ más largo de las claves de } d)$

**Descripción:** devuelve **true** si y sólo  $k$  está definido en el diccionario.

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

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

**Post**  $\equiv \{res = \text{Significado}(d, k)\}$

**Complejidad:**  $\Theta(\text{longitud del elemento de tipo } \kappa \text{ más largo de las claves de } d)$

**Descripción:** devuelve el significado de la clave  $k$  en  $d$ .

**Aliasing:**  $res$  es modificable si y sólo si  $d$  es modificable.