

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 $\equiv \{s \neq ll \wedge \text{todosEnRango}(p \cup f \cup ch \cup \{s, ll\}, \text{ancho}, \text{alto}) \wedge \{s, ll\} \cap (f \cup p) = \emptyset \wedge \text{disjuntosDeAPares}(p, f, ch)\}$
Post $\equiv \{\text{res} = \text{nuevoMapa}(\text{ancho}, \text{alto}, s, ll, p, f, ch)\}$
Complejidad: $\Theta(\text{alto} \times \text{ancho})$
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 $\equiv \{\text{res} = \text{inicio}(m)\}$
Complejidad: $\Theta(1)$
Descripción: Devuelve la salida del mapa

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

FANTASMAS(**in** *m*: mapa) \rightarrow *res* : arreglo(arreglo(bool))
Pre $\equiv \{\text{true}\}$
Post $\equiv \{(\forall c : \text{coordenada})(\text{enRango}(c, \text{largo}(m), \text{alto}(m)) \Rightarrow_{\text{L}} (c \in \text{fantasmas}(m) \iff \text{res}[c] = \text{true}))\}$ Faltaría hablar del tamaño de la matriz.
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 $\equiv \{(\forall c : \text{coordenada})(\text{enRango}(c, \text{largo}(m), \text{alto}(m)) \Rightarrow_{\text{L}} (c \in \text{paredes}(m) \iff \text{res}[c] = \text{true}))\}$
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 $\text{tupla}(\text{alto: nat} , \text{ancho: nat} , \text{salida: coordenada} , \text{llegada: coordenada} , \text{chocolates: arreglo}(\text{arreglo}(\text{bool})) , \text{fantasmas: arreglo}(\text{arreglo}(\text{bool})) , \text{paredes: arreglo}(\text{arreglo}(\text{bool})) , \text{chocosOri: conj}(\text{coordenada}))$

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

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

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. y son distintos
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 $m.\text{paredes}$.

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

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

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



Algoritmos

crearMapa(*in* $\text{alto}: \text{nat}$, *in* $\text{ancho}: \text{nat}$, *in* $s: \text{coordenada}$, *in* $ll: \text{coordenada}$, *in* $f: \text{conj}(\text{coordenada})$,
in $p: \text{conj}(\text{coordenada})$, *in* $ch: \text{conj}(\text{coordenada})$) $\rightarrow res: \text{mp}$

```

1:  $res.\text{alto} \leftarrow alto$   $\triangleright \Theta(1)$ 
2:  $res.\text{ancho} \leftarrow ancho$   $\triangleright \Theta(1)$ 
3:  $res.\text{salida} \leftarrow s$   $\triangleright \Theta(1)$ 
4:  $res.\text{llegada} \leftarrow ll$   $\triangleright \Theta(1)$ 
5:  $res.\text{chocosOri} \leftarrow ch$   $\triangleright \Theta(c)$ 
6:  $res.\text{fantasmas} \leftarrow \text{inicializar}(\text{alto}, \text{inicializar}(\text{ancho}, \text{false}))$   $\triangleright \Theta(\text{ancho} \times \text{alto})$ 
7:  $res.\text{paredes} \leftarrow \text{inicializar}(\text{alto}, \text{inicializar}(\text{ancho}, \text{false}))$   $\triangleright \Theta(\text{ancho} \times \text{alto})$ 
8:  $res.\text{chocolates} \leftarrow \text{inicializar}(\text{alto}, \text{inicializar}(\text{ancho}, \text{false}))$   $\triangleright \Theta(\text{ancho} \times \text{alto})$ 
9:  $itConj(\text{coordenada}) \text{ itF} \leftarrow \text{crearIt}(f)$   $\triangleright \Theta(1)$ 
10: while  $\text{haySiguiente}(\text{itF})$  do  $\triangleright \Theta(1)$ . El ciclo se hace  $\#f$  veces
11:    $res.\text{fantasmas}[\text{siguiente}(\text{itF})] \leftarrow \text{true}$   $\triangleright \Theta(1)$ 
12:    $\text{avanzar}(\text{itF})$   $\triangleright \Theta(1)$ 
13: end while
14:  $itConj(\text{coordenada}) \text{ itP} \leftarrow \text{crearIt}(p)$   $\triangleright \Theta(1)$ 
15: while  $\text{haySiguiente}(\text{itP})$  do  $\triangleright \Theta(1)$ . El ciclo se hace  $\#p$  veces
16:    $res.\text{paredes}[\text{siguiente}(\text{itP})] \leftarrow \text{true}$   $\triangleright \Theta(1)$ 
17:    $\text{avanzar}(\text{itP})$   $\triangleright \Theta(1)$ 
18: end while
19:  $itConj(\text{coordenada}) \text{ itCh} \leftarrow \text{crearIt}(ch)$   $\triangleright \Theta(1)$ 
20: while  $\text{haySiguiente}(\text{itCh})$  do  $\triangleright \Theta(1)$ . El ciclo se hace  $c$  veces
21:    $res.\text{chocolates}[\text{siguiente}(\text{itCh})] \leftarrow \text{true}$   $\triangleright \Theta(1)$ 
22:    $\text{avanzar}(\text{itCh})$   $\triangleright \Theta(1)$ 
23: end while

```

Complejidad: $\Theta(\text{alto} \times \text{ancho})$, ya que $\#f, \#p$ y $c \leq \text{alto} \times \text{ancho}$

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

```

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

```

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 puntero (no modificable)

Referencia

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 puntero (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 puntero (modificable)

Servicios Usados

Del módulo Conjunto Lineal(α) se utilizó:

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

Del módulo Arreglo(α) se utilizó:

- $\bullet[\bullet]$: 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

No es óptimo, pero vale :)

MOVERSE(**in/out** $p : \text{partida}$, **in** $d : \text{dir}$)
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** $p : \text{partida}$) $\rightarrow res : \text{coordenada}$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res = jugador(p)\}$
Complejidad: $\Theta(1)$
Descripción: Retorna la coordenada actual del jugador
Aliasing: No aplica

GANO?(**in** $p : \text{partida}$) $\rightarrow res : \text{bool}$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res = gano?(p)\}$
Complejidad: $\Theta(1)$

Descripción: Retorna si el jugador ganó la partida

Aliasing: No aplica

$\text{PERDIO?}(\text{in } p : \text{partida}) \rightarrow \text{res} : \text{bool}$

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

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

Complejidad: $\Theta(1)$

Descripción: Retorna si el jugador perdió la partida

Aliasing: No aplica

$\text{PUNTAJE}(\text{in } p : \text{partida}) \rightarrow \text{res} : \text{nat}$

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

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

Complejidad: $\Theta(1)$

Descripción: Retorna el puntaje de la partida

Aliasing: No aplica

Representación

partida se representa con estrPartida

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

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

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

1. $0 \leq \text{inmunidad} \leq 10.$ * $\text{puntaje} \leq \text{distManhattan}(\text{actual}, \text{inicio}(\text{mapa}))$
2. *actual* debe ser una coordenada válida.

$\text{Abs} : \text{estrPartida } e \rightarrow \text{partida}$

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

Algoritmos

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

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

Complejidad: $\Theta(1)$

res.mapa se le asigna *m* por referencia

```

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:     if  $\text{chocolates}(m)[p.\text{actual}]$  then  $\triangleright \Theta(1)$ 
18:        $p.\text{inmunidad} \leftarrow 10$   $\triangleright \Theta(1)$ 
19:        $\text{chocolates}(m)[p.\text{actual}] \leftarrow \text{false}$   $\triangleright \Theta(1)$ 
20:     else
21:        $p.\text{inmunidad} \leftarrow p.\text{inmunidad} - 1$   $\triangleright \Theta(1)$ 
22:     end if
23:   end if
24: end if
25:  $\text{res.puntaje} \leftarrow 0$   $\triangleright \Theta(1)$ 

```

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)  $\rightarrow \text{res}$  : bool
1:  $\text{res} \leftarrow \text{fantasmas}(p.\text{mapa})[p.\text{actual}] \wedge p.\text{inmunidad} == 0$   $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(1)$ ↘ ¿Y la distancia de asustarse?

puntaje(in $p : \text{partida}$) $\rightarrow res : \text{nat}$

1: $res \leftarrow p.puntaje$

$\triangleright \Theta(1)$

Complejidad: $\Theta(1)$

Servicios Usados

Del módulo $\text{Arreglo}(\alpha)$ 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 una lista ordenada desde mejor puesto a peor puesto, con el nombre y el mejor puntaje de cada jugador que haya ganado alguna partida

Aliasing: Se retorna una referencia no modificable

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(mapa: mapa , partida: partida , jugando: bool , jugador: string , jugadores: diccTrie(string, itLista(<string, nat>)) , ranking: lista(<jugador:string, puntos: nat>) , puesto: itLista(<string, nat>))`

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

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

1. Todos los iteradores de los significados de *jugadores* apuntan a un elemento de la lista *ranking*, cuya primera componente es su clave.
2. Todos los elementos de la lista *ranking* son apuntados por exactamente un significado de *jugadores*.
3. Recorriendo la lista *ranking* en orden, la segunda componente tiene que ir en orden creciente.
4. El mapa de *partida* debe ser *mapa*.
5. Si *jugando* es true, entonces *jugador* no es vacío y *partida* no puede estar ganada ni perdida.
6. *puesto* apunta a un elemento de *ranking*.

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

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

Técnicamente no existe. Charlar.

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

Algoritmos

crearFichin(in *m* : mapa) → *res* : fichin

1: <i>res.mapa</i> ← <i>m</i>	▷ Θ(1)
2: <i>res.partida</i> ← <i>crearPartida</i> (<i>m</i>)	▷ Θ(1)
3: <i>res.jugando</i> ← <i>false</i>	▷ Θ(1)
4: <i>res.jugador</i> ← ""	▷ Θ(1)
5: <i>res.jugadores</i> ← <i>vacío</i> ()	▷ Θ(1)
6: <i>res.ranking</i> ← <i>vacío</i> ()	▷ Θ(1)
7: <i>res.puesto</i> ← <i>NULL</i>	▷ Θ(1)

Complejidad: Θ(1)

A *res.mapa* se le asigna una referencia a *m*

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

1: $f.\text{mapa} \leftarrow \text{resetearChocos}(f.\text{mapa})$	$\triangleright \Theta(c)$
2: $f.\text{partida} \leftarrow \text{crearPartida}(f.\text{mapa})$	$\triangleright \Theta(1)$
3: $f.\text{jugando} \leftarrow \text{true}$	$\triangleright \Theta(1)$
4: $f.\text{jugador} \leftarrow j \quad O(J)$	$\triangleright \Theta(1)$
5: $f.\text{puesto} = \text{crearIt}(f.\text{ranking})$	$\triangleright \Theta(1)$

Complejidad: $\Theta(c)$

Esto tiene que apuntar al siguiente mejor puntaje respecto del jugador que está jugando. No se mueve durante la partida.

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

1: $\text{move}(f.\text{partida}, d)$	$\triangleright \Theta(1)$
2: if $\text{HaySiguiente}(f.\text{puesto}) \wedge_L \text{puntaje}(f.\text{partida}) \geq \text{puntos}(\text{Siguiente}(f.\text{puesto}))$ 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 $\text{puntos}(\text{Siguiente}(\text{significado}(f.\text{jugadores}, f.\text{jugador}))) > \text{puntaje}(f.\text{partida})$ then	$\triangleright \Theta(J)$
8: $\text{EliminarSiguiente}(\text{significado}(f.\text{jugadores}, f.\text{jugador}))$	$\triangleright \Theta(J)$
9: $\text{AgregarComoSiguiente}(f.\text{puesto}, \langle f.\text{jugador}, \text{puntaje}(f.\text{partida}) \rangle)$	$\triangleright \Theta(1)$
10: $\text{definir}(f.\text{jugadores}, f.\text{jugador}, f.\text{puesto})$	$\triangleright \Theta(J)$
11: end if	
12: else	
13: $\text{AgregarComoSiguiente}(f.\text{puesto}, \langle f.\text{jugador}, \text{puntaje}(f.\text{partida}) \rangle)$	$\triangleright \Theta(1)$
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 puntero no modificable

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

1: if $\text{HayAnterior}(\text{significado}(f.\text{jugadores}, f.\text{jugador}))$ then	$\triangleright \Theta(J)$
2: $res \leftarrow \text{Anterior}(\text{significado}(f.\text{jugadores}, f.\text{jugador}))$	$\triangleright \Theta(J)$
3: else	
4: $res \leftarrow \text{Siguiente}(\text{significado}(f.\text{jugadores}, f.\text{jugador}))$	$\triangleright \Theta(J)$
5: end if	

Complejidad: $\Theta(|J|)$

Servicios Usados

Del módulo Lista Enlazada(α) se utilizó:

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

El módulo Diccionario Trie (κ, σ) es un diccionario implementado sobre un trie con la siguiente interfaz:

se explica con: DICCIONARIO.

géneros: diccTrie.

DEFINIR(in/out d : dicc(κ, σ), in k : κ , in s : σ)

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

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

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

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 : dicc(κ, σ), in k : κ) $\rightarrow res$: 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 : dicc(κ, σ), in k : κ) $\rightarrow res$: σ

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.