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

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

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

Rep : estrMapa $\longrightarrow$ bool
Rep(m) $\equiv$ true $\Longleftrightarrow$

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

Abs : estrMapa e $\longrightarrow$ mapa {Rep(e)}

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

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)`) $\rightarrow$ *res* : mp

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

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

---

resetearChocos(**in/out** *m* : `mp`)

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

Complejidad: $\Theta(c)$

---

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

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

Complejidad: $\Theta(1)$

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

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

Complejidad: $\Theta(1)$

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

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

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

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

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

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

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

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

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

Servicios Usados

Del módulo Conjunto Lineal(α) se utilizó:

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

Del módulo Arreglo(α) 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 : `nat`,**in** $a : \alpha$) $\rightarrow$ res : `arreglo(`α`)`
Pre $\equiv$ {true}
Post $\equiv$ $\{long(res) = dim \wedge (\forall i : nat)(0 \leq i < dim \Rightarrow_{\text{L}} res[i] == a)\}$
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 : `mapa`) $\rightarrow$ res : `partida`
Pre $\equiv$ {true}
Post $\equiv$ {res = nuevaPartida(m)}
Complejidad: $\Theta(1)$
Descripción: Crea una nueva instancia de partida
Aliasing: Mapa es pasado por referencia modificable

MOVERSE(**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 $\longrightarrow$ bool
Rep(p) $\equiv$ true $\Longleftrightarrow$

1. $0 \leq inmunidad \leq 10$.
2. *actual* debe ser una coordenada válida.

Abs : estrPartida e $\longrightarrow$ partida {Rep(e)}
Abs(e) $\equiv$ p: partida | $mapa(p) = e.mapa \wedge jugador(p) = e.actual \wedge cantMov(p) = e.puntaje \wedge inmunidad(p) = e.inmunidad \wedge_{\text{L}} (\forall c : coordenada)(c \in chocolates(p) \Longleftrightarrow chocolates(e.mapa)[c] == true)$

Algoritmos

---

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

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

Complejidad: $\Theta(1)$
res.mapa se le asigna m por referencia

---

moverse(**in/out** p : `partida`,**in** d : `dir`)

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

Complejidad: $\Theta(1)$

---

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

1: $res \leftarrow p.actual$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$

---

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

1: $res \leftarrow p.actual == llegada(p.mapa)$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$

---

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

1: $res \leftarrow fantasmas(p.mapa)[p.actual] \land p.inmunudad == 0$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$

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

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

Complejidad: $\Theta(1)$

Servicios Usados

Del módulo Arreglo(α) 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 : `mapa`) $\rightarrow$ res : `fichin`
Pre $\equiv$ {true}
Post $\equiv$ {res= nuevoFichin(m)}
Complejidad: $\Theta(1)$
Descripción: Crea una nueva instancia de fichin
Aliasing: Mapa es pasado por referencia modificable

NUEVAPARTIDA(**in/out** f : `fichin`, **in** $string$: `j`)
Pre $\equiv$ $\{f = F_0 \wedge \neg(alguienJugando?(f))\}$
Post $\equiv$ $\{f = nuevaPartida(F_0, j)\}$
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 : `fichin`, **in** dir : `d`)
Pre $\equiv$ $\{f = F_0 \wedge alguienJugando?(f)\}$
Post $\equiv$ $\{f = mover(F_0, d)\}$
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 : `fichin`) $\rightarrow$ res : `lista(`$\langle$`string,nat`$\rangle$`)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = ranking(f)\}$
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 modicable

OBJETIVO(**in** f : `fichin`) $\rightarrow$ res : $\langle$`string,nat`$\rangle$
Pre $\equiv$ $\{alguienJugando?(f) \wedge_{\text{L}} definido(ranking(f), jugadorActual(f))\}$
Post $\equiv$ $\{res = objetivo(f)\}$
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⟩)`)

Rep : estrFichin $\longrightarrow$ bool
Rep(f) $\equiv$ true $\Longleftrightarrow$

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 ls lista *ranking* son apuntados por exactamente un significado de *jugadores*.
3. Recorriendo la lista *rankimg* 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*.

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

Algoritmos

---

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

1: $res.mapa \leftarrow m$ ▷ $\Theta(1)$
2: $res.partida \leftarrow crearPartida(m)$ ▷ $\Theta(1)$
3: $res.jugando \leftarrow false$ ▷ $\Theta(1)$
4: $res.jugador \leftarrow$ "" ▷ $\Theta(1)$
5: $res.jugadores \leftarrow vacio()$ ▷ $\Theta(1)$
6: $res.ranking \leftarrow vacio()$ ▷ $\Theta(1)$
7: $res.puesto \leftarrow NULL$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$
A $res.mapa$ se le asigna una referencia a m

---

nuevaPartida(**in/out** f : `fichin`, **in** j : `string`)

1: $f.mapa \leftarrow resetearChocos(f.mapa)$ ▷ $\Theta(c)$
2: $f.partida \leftarrow crearPartida(f.mapa)$ ▷ $\Theta(1)$
3: $f.jugando \leftarrow true$ ▷ $\Theta(1)$
4: $f.jugador \leftarrow j$ ▷ $\Theta(1)$
5: $f.puesto =$ crearIt($f.ranking$) ▷ $\Theta(1)$

Complejidad: $\Theta(c)$

---

mover(**in/out** f : `fichin`, **in** d : `dir`)

1: $moverse(f.partida, d)$ ▷ $\Theta(1)$
2: **if** HaySiguiente($f.puesto$) $\wedge_{\text{L}}$ puntaje($f.partida$) $\geq$ puntos(Siguiente($f.puesto$)) **then** ▷ $\Theta(1)$
3: Avanzar($f.puesto$) ▷ $\Theta(1)$
4: **end if**
5: **if** gano?($f.partida$) **then** ▷ $\Theta(1)$
6: **if** definido?($f.jugadores, f.jugador$) **then** ▷ $\Theta(|J|)$
7: **if** puntos(Siguiente(significado($f.jugadores, f.jugador$))) $>$ puntaje($f.partida$) **then** ▷ $\Theta(|J|)$
8: EliminarSiguiente(significado($f.jugadores, f.jugador$)) ▷ $\Theta(|J|)$
9: AgregarComoSiguiente($f.puesto, \langle f.jugador,$ puntaje($f.partida$)$\rangle$) ▷ $\Theta(1)$
10: definir($f.jugadores, f.jugador, f.puesto$) ▷ $\Theta(|J|)$
11: **end if**
12: **else**
13: AgregarComoSiguiente($f.puesto, \langle f.jugador,$ puntaje($f.partida$)$\rangle$) ▷ $\Theta(1)$
14: definir($f.jugadores, f.jugador, f.puesto$)) ▷ $\Theta(|J|)$
15: **end if**
16: **end if**
17: $f.jugando \leftarrow \neg$ (gano?($f.partida$) $\vee$ perdio?($f.partida$)) ▷ $\Theta(1)$

Complejidad: $\Theta(|J|)$

---

verRanking(**in** f : `fichin`) $\rightarrow$ res : lista($\langle$string,nat$\rangle$)

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

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

---

objetivo(**in** f : `fichin`) $\rightarrow$ res : $\langle$string,nat$\rangle$

1: **if** HayAnterior(significado($f.jugadores, f.jugador$)) **then** ▷ $\Theta(|J|)$
2: $res \leftarrow$ Anterior(significado($f.jugadores, f.jugador$)) ▷ $\Theta(|J|)$
3: **else**
4: $res \leftarrow$ Siguiente(significado($f.jugadores, f.jugador$)) ▷ $\Theta(|J|)$
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** $k: \kappa$, **in** $s: \sigma$)
Pre $\equiv \{d = D_0\}$
Post $\equiv \{d = \text{definir}(D_0, k, s)\}$
Complejidad: Θ(longitud del elemento de tipo κ 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** $k: \kappa$) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv \{res = \text{def?}(d, k)\}$
Complejidad: Θ(longitud del elemento de tipo κ 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** $k: \kappa$) $\rightarrow$ res : σ
Pre $\equiv$ {def?(d, k)}
Post $\equiv \{res = \text{Significado}(d, k)\}$
Complejidad: Θ(longitud del elemento de tipo κ 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.