Algoritmos y Estructuras de Datos II

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

Trabajo Práctico 3: "Pacalgo2"

Grupo: tomarAgua()

Integrante	LU	Correo electrónico
Reyna Maciel, Guillermo José	393/20	guille.j.reyna@gmail.com
Casado Farall, Joaquin	072/20	joakinfarall@gmail.com
Fernández Spandau, Luciana	368/20	fernandezspandau@gmail.com
Chumacero, Carlos Nehemias	492/20	chumacero2013@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Renombres

Coordenada **es** tupla(nat, nat)
Tablero **es** array(array(tupla(bool, bool, bool)))
Jugador **es** string

2. Módulo: Fichín

Interfaz

Especificación de las operaciones auxiliares utilizadas en la interfaz

se explica con: FICHÍN

géneros: `fichin`.

Operaciones básicas de fichin

NUEVOFICHIN(**in** m : `mapa`) $\rightarrow$ res : `fichin`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ nuevoFichin(m)}
Complejidad: $\Theta(\#chocolates + \#paredes + \#fantasmas)$
Descripción: genera una instancia del Fichin.

NUEVAPARTIDA(**in/out** f : `fichin`, **in** j : `jugador`)
Pre $\equiv$ {f = $f_0 \wedge \neg$alguienJugando?(f_0)}
Post$\equiv$ {f $=_{\text{obs}}$ nuevaPartida(f_0, j}
Complejidad:$\Theta(c)$
Descripción: Crea una partida para el jugador especificado, en este proceso tambien se restauran los chocolates en el mapa.

MOVER(**in/out** f : `fichin`, **in** d : `direccion`)
Pre $\equiv$ {f = $f_0 \wedge \neg$alguienJugando?(f_0)}
Post$\equiv$ {f $=_{\text{obs}}$ nuevaPartida(f_0, d)}
Complejidad:$\Theta(1)$ normalmente / $\Theta(|J|)$ cuando ganó o perdió.
Descripción: Realiza el movimiento del jugador.

MAPA(**in** f : `fichin`) $\rightarrow$ res : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ mapa(f)}
Complejidad: $\Theta(1)$
Descripción: Devuelve el mapa actual.
Aliasing: Se devuelve una referencia inmutable.

ALGUIENJUGANDO?(**in** f : `fichin`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ alguienJugando?(f)}
Complejidad: $\Theta(1)$
Descripción: Devuelve true si hay alguien jugando, de lo contrario devuelve false

JUGADORACTUAL(**in** f : `fichin`) $\rightarrow$ res : `jugador`
Pre $\equiv$ {alguienJugando?(f)}
Post $\equiv$ {$res =_{\text{obs}}$ jugadorActual(f)}
Complejidad: $\Theta(1)$
Descripción: Devuelve el nombre del jugador actual
Aliasing: Se devuelve una referencia inmutable.

PARTIDAACTUAL(**in** f : `fichin`) $\rightarrow$ res : `partida`
Pre $\equiv$ {alguienJugando?(f)}
Post $\equiv$ {$res =_{\text{obs}}$ jugadorActual(f)}

Complejidad: $\Theta(1)$
Descripción: Devuelve una referencia a la partida actual
Aliasing: Se devuelve una referencia inmutable.

RANKING(**in** f : `fichin`) $\rightarrow$ res : `ranking`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ ranking(f)}
Complejidad: $\Theta(1)$
Descripción: Devuelve el diccionario usado en el fichin
Aliasing: Se devuelve una referencia inmutable.

OBJETIVO(**in** f : `fichin`) $\rightarrow$ res : `tupla(jugador, nat)`
Pre $\equiv$ {alguienJugando?(f)}
Post $\equiv$ {$res =_{\text{obs}}$ objetivo(f, t)}
Complejidad: $\Theta(\#J.|J|)$
Descripción: Devuelve solo si existe una tupla con el nombre y puntaje del jugador que supera inmediatamente en el ranking al jugador actual.

Representación

`fichin` se representa con `estr`

donde `estr` es `tupla`(m: `puntero(mapa)`, p: `puntero(partida)`, *tablero*: `array(array(tupla(bool, bool, bool)))` , *hayAlguien*: `bool` , *jugador*: `string` , *ranking*: `diccTrie(string, nat)`)

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\iff$ (1) $\wedge$ (2)
donde:
(1) $\equiv$ e.m = (e.p $\rightarrow$ *mapa*)
(2) $\equiv$ $(\forall i, j$: nat$)((0 \leq i < e.m.largo \wedge 0 \leq j < e.m.alto) \Rightarrow (\pi_0$(e.tablero[i][j]) $\Leftrightarrow$ tupla(i, j) $\in$ e.m.paredes)) $\wedge_L$
$(\forall i, j$: nat$)((0 \leq i < e.m.largo \wedge 0 \leq j < e.m.alto) \Rightarrow (\pi_1$(e.tablero[i][j]) $\Leftrightarrow$ tupla(i, j) $\in$ e.m.fantasmas)) $\wedge_L$
$(\forall i, j$: nat$)((0 \leq i < e.m.largo \wedge 0 \leq j < e.m.alto) \Rightarrow (\pi_2$(e.tablero[i][j]) $= true \Rightarrow$ tupla(i, j) $\in$ e.m.chocolates))
en palabras:
m debe ser igual al mapa de p (partida)
tablero debe ser igual al mapa (menos los chocolates que dejamos en falso)

Abs : estr e $\longrightarrow$ Fichín {Rep(e)}
Abs(e) $\equiv$ $(\forall e : estr) Abs(e) =_{\text{obs}} f : fichín | (1) \wedge (2) \wedge (3) \wedge (4) \wedge (5)$
(1) $\equiv e.m =_{\text{obs}} mapa(f)$
(2) $\equiv e.p =_{\text{obs}} partida(f)$
(3) $\equiv e.hayAlguien =_{\text{obs}} alguienJugando?(f)$
(4) $\equiv e.jugador =_{\text{obs}} jugadorActual(f)$
(5) $\equiv e.ranking =_{\text{obs}} ranking(f)$

Algoritmos

---

iNuevoFichín(**in** m : `puntero(mapa)`)$\rightarrow$ res : estr
res.mapa $\leftarrow$ m
res.tablero $\leftarrow$ inicializarTablero(m) ▷ $\mathcal{O}(c+p+f)$
res.partida $\leftarrow$ null
res.enCurso $\leftarrow$ false
res.ranking $\leftarrow$ Vacio()
res.jugador $\leftarrow$ ""
Complejidad: $\mathcal{O}(c+p+f)$, donde $c = \#chocolates$, $p = \#paredes$, $f = \#fantasmas$

---

iNuevaPartida(in/out f : `estr`, **in** j : *string*)

RepoblarChocolates(f.tablero, f.m)
f.p ← NuevaPartida(f.m, f.tablero)
f.hayAlguien ← true
f.jugador ← j
Complejidad: $\Theta(c)$
Justificación: **Cuando se repueblan los chocolates se hace en** $\Theta(c)$ **dado a que se recorre un arreglo con las coordenadas de estas para restaurar los chocolates. Esto sumado a las operaciones en** $\Theta(1)$ **que le siguen dan como resultado a una complejidad** $\Theta(c)$

---

iMover(in/out f : `estr`, **in** d: `dirección`)

Mover(f→p, d)
if Ganó?(f→p) ∧ (Def?(f.jugador, f.ranking) ∧ CantMov(f→p) < f.ranking[f.jugador])
∨ (¬ Def?(f.jugador, f.ranking)) **then**
    Definir(f.ranking, f.jugador, CantMov(f→p))
    f.hayAlguien → false
else
    if SeAsustó?(f→p, d) **then**
        f.hayAlguien ← false
    end if
end if
Complejidad: $\theta(1)$ si no ganó, lo único que se hace es Mover(partida, dirección) y eso cuesta $\theta(1)$. Sino, hay que buscarlo en el ranking y eso cuesta $\theta(|J|)$.

---

iMapa(in f : `estr`) → res : *puntero*(*mapa*)

res ← f.m
Complejidad: $\Theta(1)$

---

iAlguienJugando?(in f **:** `estr`**)**→ res: bool

res ← f.hayAlguien
Complejidad: $\theta(1)$

---

iJugadorActual(in f : `estr`) → res : *string*

res ← f.jugador
Complejidad: $\Theta(1)$

---

iPartidaActual?(in f **:** `estr`**)**→ res: partida

res ← f→partida
Complejidad: $\theta(1)$

---

iRanking(in f : `estr`) → res : *ranking*

res ← f.ranking
Complejidad: $\Theta(1)$

Objetivo(in f : `estr`)→ res: tupla(jugador, nat)

Recorre el Trie del ranking, se queda con el jugador cuyo puntaje es el siguiente mayor al del jugador actual (el que está jugando). Sino no hay otro jugador, se queda con el actual. Devuelve en una tupla el nombre del jugador y su puntaje.

Complejidad: $\theta(\#J \cdot |J|)$

---

iInicializarTablero(**in** m : `mapa`) → res : array(array(tupla(bool, bool, bool)))

t ← arreglo[largo(m)] de arreglo[alto(m)] de tupla ⟨bool, bool, bool⟩(false)
for c in chocolates(m) **do** ▷ $\theta(c)$
  $\pi_2(t[\pi_0(c)][\pi_1(c)])$← true
end for
for p in paredes(m) **do** ▷ $\theta(p)$
  $\pi_0(t[\pi_0(p)][\pi_1(p)])$ ← true
end for
for f in fantasmas(m) **do** ▷ $\theta(f)$
  $\pi_1(t[\pi_0(f)][\pi_1(f)])$ → true
end for
Complejidad: $\Theta(c) + \Theta(p) + \Theta(f)$

---

iRepoblarChocolates(**in/out** t : `tablero` m **in** m : `mapa`)

for c in Chocolates(m) **do**
  $\pi_2(t[\pi_0(c)][\pi_1(c)]) \leftarrow true$
end for
Complejidad: $\Theta(c)$

---

3. Módulo: Partida

Interfaz

NuevaPartida(**in** m : `mapa`, **in** t : `tablero`) → res : `estr`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}}$ NuevaPartida(m)}
Complejidad: $\mathcal{O}(1)$
Descripción: genera una nueva partida.
Aliasing: Tanto el mapa como el tablero son pasados por referencia y la partida

Mover(**in/out** p: `estr`, **in** d: `direccion`)
Pre ≡ {$p =_{\text{obs}} p_0 \wedge \neg($ gano?$(p) \vee$ perdio?(p) $)$}
Post ≡ {$res =_{\text{obs}}$ Mover(p_0, d)}
Complejidad: $\mathcal{O}(1)$
Descripción: mueve la coordenada actual del jugador en la direccion indicada en el parametro.

Mapa(**in** p: `estr`) → res : `mapa`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}}$ Mapa(p)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve el mapa de la partida
Aliasing: Devuelve una referencia inmutable.

Jugador(**in** p: `estr`) → res : `coordenada`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}}$ Jugador(p)}
Complejidad: $\mathcal{O}(1)$

Descripción: Devuelve la coordenada actual del jugador.
Aliasing: Devuelve la coordenada por copia.

CantMov(**in** p: `estr`) $\rightarrow$ *res* : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {*res* $=_{\text{obs}}$ CantMov(p)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve la cantidad de movimientos que ha realizado el jugador en la partida.
Aliasing: Devuelve la cantidad de movimientos por copia.

Inmunidad(**in** p: `estr`) $\rightarrow$ *res* : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {*res* $=_{\text{obs}}$ Inmunidad(p)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve la cantidad de movimientos con inmunidad restantes.
Aliasing: Devuelve la cantidad de movimientos por copia

Gano?(**in** p: `estr`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {*res* $=_{\text{obs}}$ Gano?(p)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve true sii la partida termino y el jugador logro llegar al punto de llegada.
Aliasing: Devuelve el booleano por copia.

Perdio?(**in** p: `estr`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {*res* $=_{\text{obs}}$ Perdio?(p)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve true sii la partida termino y el jugador se encuentra en una posición en la cual es asustado por un fastasma.
Aliasing: Devuelve el booleano por copia.

Representación

`Partida` **se representa con** `estr`

donde `estr` es `tupla`(*mapa*: `puntero(mapa)`,
tablero: `puntero(tablero)` ,
coordenadaActual: `coordenada` ,
cantMov: `nat`,
movsInmunes: `nat`,
perdio?: `bool`,
gano?: `bool`)

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\iff$ (1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4) $\wedge$ (5) $\wedge$ (6)
donde:
(1) $\equiv$ $\neg$(e.gano? $\vee$ e.perdio?) $\vee$ (e.gano? = $\neg$e.perdio?)
(2) $\equiv$ (0 $\leq$ e.coordenadaActual.first $<$ e.mapa.largo) $\wedge$ (0 $\leq$ e.coordenadaActual.second $<$ e.mapa.alto)
(3) $\equiv$ long(e.tablero) = e.mapa.largo $\wedge$ long(e.tablero[0]) = e.mapa.alto $\wedge$ ($\forall c$: *coordenada*)(($c \in$ e.mapa.chocolates $\Rightarrow$ e.tablero[c.first][c.second].third) $\vee$ (c $\in$ e.mapa.fantasmas $\iff$ e.tablero[c.first][c.second].second $\vee$ (c $\in$ e.mapa.paredes $\iff$ e.tablero[c.first][c.second].first))
(4) $\equiv$ e.gano? $\iff$ e.coordenadaActual $=_{\text{obs}}$ e.mapa.llegada
(5) $\equiv$ e.perdio? $\iff$ (seAsusta?(e.mapa, e.tablero, e.coordenadaActual) $\wedge$ e.movsInmunes = 0)
(6) $\equiv$ $\neg$(e.tablero[e.coordenadaActual.first][e.coordenadaActual.second].first)

Abs : estr e $\longrightarrow$ Partida p {Rep(e)}
Abs(e) $\equiv$ (1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4) $\wedge$ (5)
(1) $\equiv$ mapa(p) $=_{\text{obs}}$ e.mapa
(2) $\equiv$ jugador(p) $=_{\text{obs}}$ e.coordenadaActual
(3) $\equiv$ ($\forall$c: coordenada)(c $\in$ chocolates(p) $\Rightarrow$ e.tablero[c.first][c.second].third)

(4) $\equiv$ CantMov(p) = e.cantMov
(5) $\equiv$ Inmunidad(p) = e.movsInmunes

Algoritmos

---

iNuevaPartida(**in** t : `mapa`, **in** t : `tablero`) $\rightarrow$ res : $estr$

$res \leftarrow < puntero(m), puntero(t), m.inicio, 0, 0, false, false >$
Complejidad: $\mathcal{O}(1)$
Justificación: Se le asignan a la estructura interna de la estr punteros hacia el mapa y el tablero y el resto de los valores (tupla, bool y nat) son por copia.

---

iMover(**in/out** p : `estr`, **in** d : `dir`)

```
if !(p.gano? —— p.perdio?) && p.esMovimientoValido(p.mapa, p.tablero, p.coordenadaActual, dir) then
    if dir = "DER" then
        p.coordenadaActual ←< p.coordenadaActual.first + 1, p.coordenadaActual.second >
    else
        if dir = "IZQ" then
            p.coordenadaActual ←< p.coordenadaActual.first − 1, p.coordenadaActual.second >
        else
            if dir = "ARR" then
                p.coordenadaActual ←< p.coordenadaActual.first, p.coordenadaActual.second + 1 >
            else
                if dir = "ABJ" then
                    p.coordenadaActual ←< p.coordenadaActual.first, p.coordenadaActual.second − 1 >
                end if
                p.cantMov ← p.cantMov + 1
                if p.movsInmunes > 0 then
                    p.movsInmunes ← p.movsInmunes − 1
                end if
                if esChocolate?(p.tablero, p.coordernadaActual) then
                    p.movsInmunes ← p.movsInmunes + 10
                    p.tablero[p.coordenadaActual.first][p.coordenadaActual.second].third ← false
                end if
                if p.movsInmunes = 0 && seAsusta?(p.mapa, p.tablero, p.coordenadaActual) then
                    p.perdio? ← true
                else
                    if p.coordenadaActual =obs p.m.llegada then
                        p.gano? ← true
                    end if
```

Complejidad: $\mathcal{O}(1)$
Justificación: Devuelve una referencia inmutable del mapa de la partida

---

iMapa(**in** p : `estr`)$\rightarrow$ res : $mapa$

$res \leftarrow p.mapa$
Complejidad: $\mathcal{O}(1)$
Justificación: Devuelve una referencia inmutable del mapa de la partida =0

---

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

$res \leftarrow p.coordenadaActual$

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

Justificación: Devuelve por copia el la coordenada correspondiente a la que se encuentra el jugador.

---

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

$res \leftarrow p.cantMov$

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

Justificación: Devuelve por copia el natural correspondiente.

---

iInmunidad(**in** p: `estr`)$\rightarrow$ res : nat

$res \leftarrow p.movsINmunes$

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

Justificación: Devuelve por copia el natural correspondiente.

---

iGano?(**in** p: `estr`)$\rightarrow$ res : $bool$

$res \leftarrow p.gano?$

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

Justificación: Devuelve por copia el bool correspondiente.

---

iPerdio?(**in** p: `estr`)$\rightarrow$ res : $bool$

$res \leftarrow p.perdio?$

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

Justificación: Devuelve por copia el bool correspondiente.

---

iesChocolate?(**in** t: `tablero`, **in** c: `coordenada`)$\rightarrow$ res : $bool$

$res \leftarrow t[c.first][c.second].third$

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

Justificación: Devuelve el bool en la posición correspondiente a si hay un chocolate en esa coordenada del mapa.

---

iesFantasma?(**in** t: `tablero`, **in** c: `coordenada`)$\rightarrow$ res : $bool$

$res \leftarrow t[c.first][c.second].second$

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

Justificación: Devuelve el bool en la posición correspondiente a si hay un fantasma en esa coordenada del mapa.

---

iesPared?(**in** t: `tablero`, **in** c: `coordenada`)$\rightarrow$ res : $bool$

$res \leftarrow t[c.first][c.second].first$

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

Justificación: Devuelve el bool en la posición correspondiente a si hay una pared en esa coordenada del mapa.

---

iesMovimientoValido?(**in** m : `mapa`, **in** t : `tablero`, **in** c : `coordenada`, **in** d : `direccion`)→ res : *bool*

if $dir = "DER"$ **then**
$res \leftarrow esPosicionValida?(m, t, < c.first + 1, c.second >)$
else
if $dir = "IZQ"$ **then**
$res \leftarrow esPosicionValida?(m, t, < c.first - 1, c.second >)$
else
if $dir = "ARR"$ **then**
$res \leftarrow esPosicionValida?(m, t, < c.first, c.second + 1 >)$
else
if $dir = "ABJ"$ **then**
$res \leftarrow esPosicionValida?(m, t, < c.first, c.second - 1 >)$
else
$res \leftarrow false$
end if

Complejidad: $\mathcal{O}(1)$
Justificación: El algoritmo devuelve true sii la posicion en la direccion pasada es una direccion valida para moverse

---

iesPosicionValida?(**in** m : `mapa`, **in** t : `tablero`, **in** c : `coordenada`)→ res : *bool*

if $(0 \leq c.first ¡ m.largo \ \&\& \ 0 \leq c.second ¡ m.alto)$ **then**
$res \leftarrow esPared(t, c)$
else
$res \leftarrow false$
end if

Complejidad: $\mathcal{O}(1)$
Justificación: El algoritmo checkea que la coordenada pasada este en rango del mapa y de ser asi devuelve el valor bool de la posicion correspondiente a si es pared la coordenada, caso contrario devuelve false. =0

---

iseAsusta?(**in** m : `mapa`, **in** t : `tablero`, **in** c : `coordenada`) → res : *bool*

$aCheckear \leftarrow << c.first + 3, c.second >, < c.first - 3, c.second >, < c.first + 2, c.second + 1 >, < c.first + 2, c.second - 1 >, < c.first - 2, c.second + 1 >, < c.first - 2, c.second - 1 >, < c.first + 1, c.second + 2 >, < c.first + 1, c.second - 2 >, < c.first - 1, c.second + 2 >, < c.first - 1, c.second - 2 >, < c.first, c.second + 3 >, < c.first, c.second - 3 >>$
$i \leftarrow 0$
while $i < 12$ **do**
if $0 \leq aCheckear[i].first < m.largo \ \&\& \ 0 \leq aCheckear[i].second < m.alto$ **then**
if $esFantasma(t, posACheckear)$ **then**
$res \leftarrow true$
end if
i $\leftarrow i + 1$ **end if**

end while
$res \leftarrow false$

Complejidad: $\mathcal{O}(1)$
Justificación: El algoritmo crea un vector con todas las posiciones en las que deberia haber un fantasma para que la coordenada pasada como parametro "se asuste", estas son 12, por lo que el ciclo correra 12 veces siempre

---

4. Módulo: mapa

Interfaz

mapa **se explica con**: MAPA

géneros: `mapa`.

Operaciones básicas de mapa

NuevoMapa(**in** *largo* : `nat`, **in** *alto* : `nat`, **in** *inicio* : `coordenada`, **in** *fin* : `coordenada`, **in** *paredes* : `conj(coordenada)`, **in** *fantasmas* : `conj(coordenada)`, **in** *chocolates* : `conj(coordenada)`) → *res* : `mapa`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ nuevoMapa(*largo*, *alto*, *inicio*, *llegada*, *paredes*, *fantasmas*, *chocolates*)}
Complejidad: Θ(*largo* × *alto*)
Descripción: genera una instancia del TAD mapa.

Largo(**in** *m* : `mapa`) → *res* : `nat`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ largo(*m*)}
Complejidad: Θ(1)
Descripción: devuelve el largo del mapa.

Alto(**in** *m* : `mapa`) → *res* : `nat`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ alto(*m*)}
Complejidad: Θ(1)
Descripción: devuelve la altura del mapa.

Inicio(**in** *m* : `mapa`) → *res* : `coordenada`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ inicio(*m*)}
Complejidad: Θ(1)
Descripción: devuelve la posición de inicio del mapa.

llegada(**in** *m* : `mapa`) → *res* : `coordenada`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ llegada(*m*)}
Complejidad: Θ(1)
Descripción: devuelve la posición de fin del mapa.

Chocolates(**in** *m* : `mapa`) → *res* : `conj(coordenada)`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ chocolates(*m*)}
Complejidad: Θ(1)
Descripción: devuelve el conjunto de posiciones de los chocolates del mapa.
Aliasing: *res* es una referencia inmutable a un conjunto de coordenadas.

Paredes(**in** *m* : `mapa`) → *res* : `conj(coordenada)`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ paredes(*m*)}
Complejidad: Θ(1)
Descripción: devuelve el conjunto de posiciones de las paredes del mapa.
Aliasing: *res* es una referencia inmutable a un conjunto de coordenadas.

Fantasmas(**in** *m* : `mapa`) → *res* : `conj(coordenada)`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ fantasmas(*m*)}
Complejidad: Θ(1)
Descripción: devuelve el conjunto de posiciones de los fantasmas del mapa.
Aliasing: *res* es una referencia inmutable a un conjunto de coordenadas.

Representación

La estructura interna del módulo mapa es una tupla de elementos que representan a los observadores del TAD

MAPA de su mismo nombre.

género se representa con `estr`

donde `estr` es `tupla`(*paredes*: `conj(coordenada)`,
fantasmas: `conj(coordenada)` ,
chocolates: `conj(coordenada)` ,
largo: `nat`,
alto: `nat`,
inicio: `coordenada`,
llegada: `coordenada`)

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\iff$ (1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4) $\wedge$ (5)
donde:
(1) $\equiv$ enRango($e, inicio$) $\wedge$ enRango($e, llegada$) $\wedge \neg(inicio = llegada)$
(2) $\equiv$ e.paredes $\cap$ e.fantasmas $\cap$ e.chocolates $= \emptyset$
(3) $\equiv$ ($\forall c$: coordenada)($c \in$ e.paredes $\cup$ e.fantasmas $\cup$ e.chocolates $\rightarrow$ enRango(e, c))
(4) $\equiv \neg(inicio, llegada \in$ e.paredes $\cup$ e.fantasmas)
(5) $\equiv$ e.largo $> 0 \wedge$ e.alto > 0

enRango : estr $\times$ coordenada $\longrightarrow$ bool
enRango(e, c) $\equiv \pi_0(c) < e.largo \wedge \pi_1(c) < e.alto$

Abs : estr e $\longrightarrow$ genero TAD {Rep(e)}
Abs(e) $\equiv$ nuevoMapa(e.largo, e.alto, e.inicio, e.llegada, e.paredes, e.fantasmas, e.chocolates)

Algoritmos

---

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

res $\leftarrow <$ *paredes*, *fantasmas*, *chocolates*, *largo*, *alto*, *inicio*, *fin* $>$
Complejidad: $\mathcal{O}(largo \times alto)$
Justificación: la complejidad va a depender del conjunto más grande entre paredes, fantasmas, y chocolates, que es como máximo de tamaño largo $\times$ alto.

---

iLargo(**in** m : `estr`) $\rightarrow$ *res* : *nat*

res $\leftarrow$ e.largo ▷ $\Theta(1)$
Complejidad: $\Theta(1)$

---

iAlto(**in** m : `estr`) $\rightarrow$ *res* : *nat*

res $\leftarrow$ e.alto ▷ $\Theta(1)$
Complejidad: $\Theta(1)$

---

iInicio(**in** m : `estr`) $\rightarrow$ *res* : *coordenada*

res $\leftarrow$ e.inicio ▷ $\Theta(1)$
Complejidad: $\Theta(1)$
Justificación: Coordenada es tupla(nat, nat). Copiar naturales es $\Theta(1)$ y la tupla tiene tamaño constante acotado.

---

iLlegada(**in** m : `estr`) $\rightarrow$ res : $coordenada$
$res \leftarrow$ e.llegada ▷ $\Theta(1)$
Complejidad: $\Theta(1)$
Justificación: Coordenada es tupla(nat, nat). Copiar naturales es $\Theta(1)$ y la tupla tiene tamaño constante acotado.

iParedes(**in** m : `estr`) $\rightarrow$ res : $conj(coordenada)$
$res \leftarrow$ e.paredes ▷ $\Theta(1)$
Complejidad: $\Theta(1)$
Justificación: res es una referencia.

iFantasmas(**in** m : `estr`) $\rightarrow$ res : $conj(coordenada)$
$res \leftarrow$ e.fantasmas ▷ $\Theta(1)$
Complejidad: $\Theta(1)$
Justificación: res es una referencia.

iChocolates(**in** m : `estr`) $\rightarrow$ res : $conj(coordenada)$
$res \leftarrow$ e.chocolates ▷ $\Theta(1)$
Complejidad: $\Theta(1)$
Justificación: res es una referencia.

Servicios usados:
Conjunto se representa por el módulo Conjunto Lineal definido en el apunte de módulos básicos. La complejidad de copy(a : conj(α)) es $|a|\times$ copy(α).

5. Módulo: Diccionario Trie

Interfaz

se explica con: Diccionario, Iterador Bidireccional.

géneros: `diccTrie`, `itDicc`.

Operaciones básicas de diccTrie

Vacío() $\rightarrow$ res : `diccTrie`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ vacio}
Complejidad: $\Theta(1)$
Descripción: genera un diccTrie vacío.

Definir(**in/out** d: `diccTrie`, **in** k: $string$, **in** s: nat)
Pre $\equiv$ {$d =_{\text{obs}} d_0$}
Post $\equiv$ {$d =_{\text{obs}}$ definir(d, k, s)}
Complejidad: $\Theta(l)$, donde l es la longitud de la clave más larga.
Descripción: define la clave k con el significado s en el diccionario. Retorna un iterador al elemento recién agregado.

Definido?(**in** d: `diccTrie`, **in** k: $string$) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ def?(d, k)}
Complejidad: $\Theta(l)$, donde l es la longitud de la clave más larga.
Descripción: devuelve `true` si y sólo k está definido en el diccionario.

SIGNIFICADO(**in** d: `diccTrie`, **in** k: *string*) $\rightarrow$ *res* : *nat*
Pre $\equiv$ {def?(d, k)}
Post $\equiv$ {alias($res =_{\text{obs}}$ Significado(d, k))}
Complejidad: $\Theta(l)$, donde l es la longitud de la clave más larga.
Descripción: devuelve el significado de la clave k en d.
Aliasing: *res* es modificable si y sólo si d es modificable.

BORRAR(**in/out** d: `diccTrie`, **in** k: *string*)
Pre $\equiv$ {$d = d_0 \wedge$ def?(d, k)}
Post $\equiv$ {$d =_{\text{obs}}$ borrar(d_0, k)}
Complejidad: $\Theta(l)$, donde l es la longitud de la clave más larga.
Descripción: elimina la clave k y su significado de d.

#CLAVES(**in** d: `diccTrie`) $\rightarrow$ *res* : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ #claves(d)}
Complejidad: $\Theta(1)$
Descripción: devuelve la cantidad de claves del diccionario.