

# Apunte de Módulos Básicos (v. 0.3 $\alpha$ )

Algoritmos y Estructuras de Datos II, DC, UBA.

1<sup>er</sup> cuatrimestre de 2019

## Índice

<b>1. Diccionario Trie (<math>\alpha</math>)</b>	<b>2</b>
<b>2. Módulo Juego</b>	<b>5</b>
<b>3. Módulo Mapa</b>	<b>9</b>
<b>4. Módulo Dirección</b>	<b>11</b>
<b>5. Módulo Acción</b>	<b>13</b>

## 1. Diccionario Trie ( $\alpha$ )

El módulo Diccionario Trie provee un diccionario básico montado sobre un trie. Solo se definen e implementan las operaciones que serán utilizadas.

### Interfaz

**parámetros formales**

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

**se explica con:**  $\text{DICCIONARIO}(\text{string}, \alpha)$ .

**géneros:**  $\text{diccTrie}(\text{string}, \alpha)$ .

### Operaciones básicas de diccionario

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

$\text{DEFINIR}(\text{in/out } d : \text{diccTrie}(\text{string}, \alpha), \text{in } k : \text{string}, \text{in } s : \alpha)$   
**Pre**  $\equiv \{d =_{\text{obs}} d_0\}$   
**Post**  $\equiv \{d =_{\text{obs}} \text{definir}(d, k, s)\}$   
**Complejidad:**  $\Theta(|k| + \text{copy}(s))$   
**Descripción:** define la clave  $k \notin \text{claves}(d)$  con el significado  $s$  en el diccionario.  
**Aliasing:** los elementos  $k$  y  $s$  se definen por copia.

$\text{DEFINIDO?}(\text{in } d : \text{diccTrie}(\text{string}, \alpha), \text{in } k : \text{string}) \rightarrow \text{res} : \text{bool}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{\text{res} =_{\text{obs}} \text{def?}(d, k)\}$   
**Complejidad:**  $\mathcal{O}(|k|)$   
**Descripción:** devuelve **true** si y sólo  $k$  está definido en el diccionario.

$\text{SIGNIFICADO}(\text{in } d : \text{diccTrie}(\text{string}, \alpha), \text{in } k : \text{string}) \rightarrow \text{res} : \sigma$   
**Pre**  $\equiv \{\text{def?}(d, k)\}$   
**Post**  $\equiv \{\text{alias}(\text{res} =_{\text{obs}} \text{obtener}(d, k))\}$   
**Complejidad:**  $\Theta(|k|)$   
**Descripción:** devuelve el significado de la clave  $k$  en  $d$ .  
**Aliasing:**  $\text{res}$  es modificable si y sólo si  $d$  es modificable.

## Representación

### Representación del diccionario

$\text{diccTrie}(\text{string}, \alpha)$  se representa con **estr**

donde **estr** es  $\text{tupla}(\text{raiz: puntero}(\text{nodo}))$

donde **nodo** es  $\text{tupla}(\text{significado: } \alpha, \text{siguientes: arreglo}(\text{puntero}(\text{nodo})) [256] )$

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

$\text{Rep}(d) \equiv \text{true} \iff \text{raiz no está contenido en ninguno de sus siguientes ni sus siguientes}$

$\text{Abs} : \text{diccTrie } d \rightarrow \text{diccTrie}(\text{string}, \alpha)$

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

$\text{Abs}(d) \equiv \text{if vacía?}(d.\text{claves}) \text{ then vacío else definir}(\text{prim}(d).\text{claves}, \text{prim}(d).\text{significado}, \text{Abs}(\text{fin}(d))) \text{ fi}$

## Algoritmos

---

---

**iVacía()**  $\rightarrow res : \text{estr}$

1: // Le asigna un nuevo nodo a la raíz  
 2:  $res \leftarrow \langle \text{raiz} : \text{nuevoNodo}() \rangle$   $\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

Justificación: La complejidad de crear un nuevo nodo es  $\Theta(1)$

---



---

---

**iSignificado(in/out d: estr, in k: string)**  $\rightarrow res : \alpha$

1:  $\text{Nodo actual} \leftarrow d.\text{raiz}$   $\triangleright \Theta(1)$   
 2: **for** ( $\text{char } c : k$ ) **do**  $\triangleright \mathcal{O}(|k|)$   
 3:      $\text{actual} \leftarrow (\text{actual} \rightarrow \text{siguientes}[\text{toInt}(c)])$   $\triangleright \Theta(1)$   
 4: **end for**  
 5:  $res \leftarrow (\text{actual} \rightarrow \text{significado})$   $\triangleright \Theta(1)$

Complejidad:  $\Theta(|k|)$

Justificación: Los accesos y las asignaciones de punteros son  $\Theta(1)$ . Como el ciclo se ejecuta  $|k|$  veces, se ejecutarán dichas asignaciones  $|k|$  veces. Luego la complejidad será  $\Theta(|k|)$ .

---



---

---

**iDefinido?(in/out d: estr, in k: string)**  $\rightarrow res : \text{bool}$

1:  $\text{Nodo actual} \leftarrow d.\text{raiz}$   $\triangleright \Theta(1)$   
 2: **for** ( $\text{char } c : k$ ) **do**  $\triangleright \mathcal{O}(|k|)$   
 3:     **if** ( $\text{actual} \rightarrow \text{siguientes}[\text{toInt}(c)] \neq \text{NULL}$ )  $\triangleright \Theta(1)$   
 4:         **then**  $\text{actual} \leftarrow (\text{actual} \rightarrow \text{siguientes}[\text{toInt}(c)])$   $\triangleright \Theta(1)$   
 5:         **else**  $res \leftarrow \text{false}$   $\triangleright \Theta(1)$   
 6:     **end if**  
 7: **end for**  
 8:  $res \leftarrow \text{true}$   $\triangleright \Theta(1)$

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

Justificación: Los accesos y las asignaciones de punteros son  $\Theta(1)$ . Como el ciclo se ejecuta a lo sumo  $|k|$  veces, se ejecutarán dichas asignaciones  $|k|$  veces como máximo. Luego la complejidad será  $\mathcal{O}(|k|)$ .

---



---

---

**iDefinir(in/out d: estr, in k: string, in s:  $\alpha$ )**

1:  $\text{Nodo actual} \leftarrow d.\text{raiz}$   
 2: **for** ( $\text{char } c : k$ ) **do**  $\triangleright \Theta(|k|)$   
 3:     // Si no tengo siguiente, lo creo  
 4:     **if** ( $\text{actual} \rightarrow \text{siguientes}[\text{toInt}(c)] == \text{NULL}$ ) **then**  $\triangleright \Theta(1)$   
 5:          $\text{actual} \rightarrow \text{siguientes}[\text{toInt}(c)] = \text{nuevoNodo}()$   $\triangleright \Theta(1)$   
 6:     **end if**  
 7:      $\text{actual} \leftarrow (\text{actual} \rightarrow \text{siguientes}[\text{toInt}(c)])$   $\triangleright \Theta(1)$   
 8: **end for**  
 9:  
 10: // Estoy parado en el nodo que va a tener el significado.  
 11: // Le asigno una copia del provisto.  
 12:  $\text{actual} \rightarrow \text{significado} \leftarrow \text{copy}(s)$   $\triangleright \Theta(\text{copy}(s))$

Complejidad:  $\Theta(|k| + \text{copy}(s))$

Justificación: Siempre se recorre toda la palabra para definirla, entonces el *for* siempre tiene  $|k|$  ciclos. La dereferenciación y comparación de punteros, e indexación en arreglos estáticos son  $\Theta(1)$ .

---

---

**inuevoNodo()**  $\rightarrow res$  : puntero(nodo) ▷ Función privada que crea un nuevo nodo

1: // Reserva la memoria para un nuevo nodo con significado null y siguientes vacíos

2:  $res \leftarrow \&\langle significado : NULL, siguientes : arreglo\_estatico[256] \text{ de } \alpha \rangle$  ▷  $\Theta(1)$

Complejidad:  $\Theta(1)$

Justificación: El tiempo de creación de un array de 255 posiciones es  $\mathcal{O}(255) \in \mathcal{O}(1)$

---

## 2. Módulo Juego

Aquí va la descripción

### Interfaz

generos: juego.

se explica con: JUEGO.

### Operaciones básicas de Juego

INICIAR(**in**  $m$ : mapa, **in**  $pjs$ : conj(jugador), **in**  $eventosFan$ : vector(evento))  $\rightarrow res$  : juego

**Pre**  $\equiv \{\neg vacio(pjs) \wedge (\forall e : evento)(est?(e, eventosFan) \Rightarrow_L e.pos \in libres(m))\}$

**Post**  $\equiv \{res =_{obs} nuevo.Juego(m, pjs, eventosFan)\}$

**Complejidad:**  $\Theta(?)$  TODO

**Descripción:** crea un nuevo juego con el mapa dado, un conjunto de jugadores, y los eventos de un fantasma.

PASARTIEMPO(**in**  $j$ : juego)  $\rightarrow res$  : juego

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

**Post**  $\equiv \{res =_{obs} pasar(j)\}$

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

**Descripción:** ejecuta un paso de tiempo cuando ningún jugador realiza una acción.

EJECUTARACCION(**in**  $j$ : juego, **in**  $a$ : accion, **in**  $pj$ : jugador)  $\rightarrow res$  : juego

**Pre**  $\equiv \{pj \in jugadores(j) \wedge_L jugadorVivo(pj, j) \wedge \neg esPasar(a)\}$

**Post**  $\equiv \{res =_{obs} step(j, a, pj)\}$

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

**Descripción:** actualiza con la acción  $a$  del jugador  $pj$ .

JUGADORESVIVOS(**in**  $j$ : juego)  $\rightarrow res$  : conj(puntero(infoPJ))

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

**Post**  $\equiv \{(\forall p : puntero(infoPJ))(p \in res \Rightarrow_L$   
 $(p \rightarrow id \in jugadores(j)) \wedge_L$   
 $(p \rightarrow vivo? \wedge jugadorVivo(p \rightarrow id, j)) \wedge$   
 $((\forall e : evento)(e \in p \rightarrow eventos \Rightarrow_L$   
 $(e.pos =_{obs} posJugador(p \rightarrow id, j)) \wedge$   
 $(e.dir =_{obs} dirJugador(p \rightarrow id, j))))\}$

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

**Descripción:** devuelve un conjunto con punteros a la información de los personajes que están vivos.

**Aliasing:** res es no modificable.

FANTASMASVIVOS(**in**  $j$ : juego)  $\rightarrow res$  : conj(infoFan)

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

**Post**  $\equiv \{fantasmaValido(j, res)\}$

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

**Descripción:** devuelve un conjunto referencias a la información de los fantasmas que están vivos.

**Aliasing:** las referencias son no modificables.

FANTASMAESPECIAL(**in**  $j$ : juego)  $\rightarrow res$  : infoFan

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

**Post**  $\equiv \{res =_{obs} fantasmaEspecial(j)\}$

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

**Descripción:** devuelve el fantasma especial.

**Aliasing:** res es una referencia no modificable.

FANTASMASVIVOSQUEDISPARAN(**in**  $j$ : juego)  $\rightarrow res$  : conj(infoFan)

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

**Post**  $\equiv \{fantasmaValido(j, res) \wedge_L$   
 $((\forall f : infoFan)(f \in res \Rightarrow_L disparando(f.eventos, step(j))))\}$

**Complejidad:**  $O(\#fv)$

**Descripción:** devuelve un conjunto con punteros a la información de los fantasmas que están vivos y disparan en el ultimo paso ejecutado en el juego.

**Aliasing:** res es un conjunto de referencias no modificables.

**VIVO?**(in  $j$ : juego, in  $pj$ : string)  $\rightarrow res$  : bool

**Pre**  $\equiv \{pj \in jugadores(j)\}$

**Post**  $\equiv \{res =_{\text{obs}} jugadorVivo(pj, j)\}$

**Complejidad:**  $O(|j|)$

**Descripción:** devuelve si un jugador está vivo

**POSOCUPADASPORDISPAROS**(in  $j$ : juego)  $\rightarrow res$  : conj(posicion)

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

**Post**  $\equiv \{res =_{\text{obs}} alcanceDisparosFantasmas(fantasmas(j), j)\}$

**Complejidad:**  $O(\#fv * m)$

**Descripción:** devuelve un conjunto de las posiciones afectadas por disparos de fantasmas en la última \*ronda\* (TODO: ronda o paso?).

Predicados auxiliares:

fantasmaValido( $j$ ,  $fs$ ):

$$(\forall f : infoFan)(f \in res \Rightarrow_L$$

$$(f.eventos \in fantasmas(j)) \wedge_L$$

$$(fantasmaVivo(f.eventos, j)) \wedge$$

$$((\forall e : evento)(e \in f.eventos \Rightarrow_L$$

$$(e.pos =_{\text{obs}} posFantasma(f.eventos, j)) \wedge$$

$$(e.dir =_{\text{obs}} dirFantasma(f.eventos, j))))$$

## Representación

### Representación de Juego

juego se representa con estr

donde  $j$  es tupla(// General

$paso$ : nat,  
 $ronda$ : nat,  
 $mapa$ : m,

// Disparos

$mapaDisparos$ : arreglo(arreglo(tupla(nat, nat))),  
 $disparosUltimoPaso$ : conj(posicion),

// Jugadores

$infoJugadores$ : diccTrie(string, infoPJ),  
 $infoActualJugadoresVivos$ : conj(infoActualPJ),  
 $infoJugadoresVivos$ : conj(puntero(infoPJ)),

// Fantasmas

$infoFantasmas$ : conj(infoFan),  
 $infoActualFantasmasVivos$ : conj(infoActualFan),  
 $infoFantasmasVivos$ : conj(itConj(infoFan)),  
 $infoFantasmaEspecial$ : itConj(infoActualFan) )

donde infoPJ es tupla( $eventos$ : vector(evento),

$vivo?$ : bool,

$infoActual$ : itConj(infoActualPJ) )

donde infoActualPJ es tupla( $identidad$ : string,

$posicion$ : pos,

$direccion$ : dir )

donde infoFan es tupla( $infoActual$ : itConj(infoActualFan),

$eventos$ : vector(evento) )

donde **infoActualFan** es **tupla**(*posicion*: **pos**,  
*direccion*: **dir** )

**Rep** : mapa  $\rightarrow$  bool  
**Rep**(*m*)  $\equiv$  true  $\iff$

**Abs** : mapa *m*  $\rightarrow$  hab {**Rep**(*m*)}  
**Abs**(*m*) =<sub>obs</sub> h: hab |

## Algoritmos

En esta sección se hace abuso de notación en los cálculos de álgebra de órdenes presentes en la justificaciones de los algoritmos. La operación de suma “+” denota secuencialización de operaciones con determinado orden de complejidad, y el símbolo de igualdad “=” denota la pertenencia al orden de complejidad resultante.

### Algoritmos del módulo

---

```

iIniciar(in m : mapa, in pjs : conj(jugador), in eventosFan : vector(evento))  $\rightarrow$  res : estr
1: // Inicializo la estructura
2: res : {
3:   // Inicializo contadores
4:   paso : 0,  $\triangleright \Theta(1)$ 
5:   ronda : 0,  $\triangleright \Theta(1)$ 
6:
7:   // Seteo el mapa
8:   mapa : m,  $\triangleright \Theta(1)$ 
9:
10:  // Inicializo el mapa de disparos con el mismo tamaño que el mapa
11:  mapaDisparos : arreglo(arreglo(tupla(nat, nat))[Tam(m)])[Tam(m)],  $\triangleright \Theta(Tam(m)^2)$ 
12:  disparosUltimoPaso : Vacio(),  $\triangleright \Theta(1)$ 
13:
14:  // Inicializo estructuras de jugadores y fantasmas como vacías
15:  infoActualJugadoresVivos : Vacio(),
16:  infoJugadoresVivos : Vacio(),
17:  infoJugadores : Vacia(),
18:  infoFantasmas : Vacio(),
19:  infoActualFantasmasVivos : Vacio(),
20:  infoFantasmasVivos : Vacia(),
21:  infoFantasmaEspecial : CrearIt(Vacio())
22: }
23:
24: // Inicializo los jugadores
25: iIniciarJugadores(res)
26:
27: // Inicializo los fantasmas
28: iIniciarFantasmas(res)

Complejidad:  $\Theta(?)$ 

```

---

**iIniciarJugadores(in j: estr)**

▷ Función privada

```

1: // Suponemos la existencia de la función
2: // dict(jugador, tupla(pos, dir)) localizarJugadores(m, conj(jugador) pjs)
3:
4: // Obtengo las posiciones y direcciones de jugadores
5: localPJs ← localizarJugadores(m, pjs)
6:
7: // Lleno las estructuras de jugadores
8: for (j, localizacion : localPJs) do
9:     // Creo la infoActual y la agrego a su conjunto
10:    infoActual ← ⟨identidad : j, posicion : localizacion.pos, direccion : localizacion.dir⟩
11:    itInfoActual ← AgregarRapido(res.infoActualJugadoresVivos, infoActual)
12:
13:    // Creo la infoPJ con la actual
14:    info ← iNuevaInfoPJ(j, localizacion, itInfoActual)
15:    // La agrego al trie y me guardo el puntero a la info guardada
16:    infoPtr ← &Definir(res.infoJugadores, j, info)
17:
18:    // Agrego al conjunto de jugadores vivos el puntero a la info del PJ
19:    AgregarRapido(res.infoJugadoresVivos, infoPtr)
20: end for

```

Complejidad:  $\Theta(?)$ **iIniciarFantasmas(in j: estr)**

▷ Función privada

```

1: // Lleno las estructuras de fantasmas
2: // Creo la infoActual y la agrego a su conjunto
3: infoActualFan ← ⟨posicion : eventosFan[0].pos, direccion : eventosFan[0].dir⟩
4: itInfoActualFan ← AgregarRapido(infoActualFan, res.infoActualFantasmasVivos)
5:
6: // Hago que el fantasma especial sea este
7: res.infoFantasmaEspecial ← itInfoActualFan
8:
9: // Creo la infoFan con la actual
10: infoFan ← ⟨infoActual : itInfoActualFan, eventos : eventosFan⟩
11: // La agrego al conjunto de información de fantasmas y me guardo su iterador
12: itInfoFan ← AgregarRapido(infoFan, res.infoFantasmas)
13:
14: // Agrego al conjunto de fantasmas vivos el iterador a la info del Fan
15: AgregarRapido(itInfoFan, res.infoFantasmasVivos)

```

Complejidad:  $\Theta(?)$ **iPasarTiempo(in j: estr)**

```

1: // Aumentas paso // Por cada fantasma // Si dispara // Agregar disparo // Agregar el disparo al conjunto
   (inteligentemente) // Agregar las pos afectadas al mapa de disparos // // Actualizo la info actual // Por cada
   jugador // Te fijas si muere // Actualizas la info actual

```

**iEjecutarAccion(in j: estr, in a: accion, in pj: jugador) → res:estr**

1:



### 3. Módulo Mapa

Aquí va la descripción

#### Interfaz

generos: mapa.

se explica con: HABITACIÓN.

#### Operaciones básicas del mapa

**NUEVOMAPA**(in  $n : \text{nat}$ )  $\rightarrow res : \text{mapa}$

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

**Post**  $\equiv \{res =_{\text{obs}} \text{nuevaHab}(n)\}$

**Complejidad**:  $\Theta(n^2)$

**Descripción**: genera un mapa de tamaño  $n \times n$ .

**OCUPAR**(in/out  $m : \text{mapa}$ , in  $c : \text{tupla}(\text{int}, \text{int})$ )

**Pre**  $\equiv \{m =_{\text{obs}} m_0 \wedge c \in \text{casilleros}(m) \wedge_L \text{libre}(m, c) \wedge \text{alcanzan}(\text{libres}(m) - c, \text{libres}(m) - c, m)\}$

**Post**  $\equiv \{m =_{\text{obs}} \text{ocupar}(c, m_0)\}$

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

**Descripción**: ocupa una posición del mapa siempre y cuando este no deje de ser conexo.

**TAM**(in  $m : \text{mapa}$ )  $\rightarrow res : \text{nat}$

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

**Post**  $\equiv \{res =_{\text{obs}} \text{tam}(m)\}$

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

**Descripción**: devuelve el tamaño del mapa.

**LIBRE**(in  $m : \text{mapa}$ , in  $c : \text{tupla}(\text{int}, \text{int})$ )  $\rightarrow res : \text{bool}$

**Pre**  $\equiv \{c \in \text{casilleros}(m)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{libre}(c, m)\}$

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

**Descripción**: devuelve si un elemento está ocupado.

#### Representación

##### Representación del mapa

El objetivo de este módulo es implementar una lista doblemente enlazada con punteros al principio y al fin. Para simplificar un poco el manejo de la estructura, vamos a reemplazarla por una lista circular, donde el siguiente del último apunta al primero y el anterior del primero apunta al último. La estructura de representación, su invariante de representación y su función de abstracción son las siguientes.

mapa se representa con  $m$

donde  $m$  es  $\text{tupla}(\text{tamano} : \text{nat}, \text{casilleros} : \text{vec}(\text{vec}(\text{bool})),)$

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

$\text{Rep}(m) \equiv \text{true} \iff \text{La longitud de } m.\text{casilleros} \text{ es igual a } \text{tamano} \wedge$

La longitud del vector  $m.\text{casilleros}$  es igual a la de todo otro vector dentro de  $\text{el} \wedge$

Es conexa

$\text{Abs} : \text{mapa } m \rightarrow \text{hab}$

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

$\text{Abs}(m) =_{\text{obs}} h : \text{hab} \mid m.\text{tamano} =_{\text{obs}} \text{tam}(h) \wedge_L$

$(\forall t : \text{tuple}(\text{nat}, \text{nat}))(0 \leq \Pi_1(t), \Pi_2(t) < m.\text{tamano} - 1 \Rightarrow_L$

$\text{libre}(m, t) =_{\text{obs}} m.\text{casilleros}[\Pi_1(t)][\Pi_2(t)])$

#### Algoritmos

En esta sección se hace abuso de notación en los cálculos de álgebra de órdenes presentes en la justificaciones de los algoritmos. La operación de suma “+” denota secuencialización de operaciones con determinado orden de complejidad,

y el símbolo de igualdad “=” denota la pertenencia al orden de complejidad resultante.

## Algoritmos del módulo

---



---

**iTam**(in  $m$ : mapa)  $\rightarrow res$ : nat

1:  $res \leftarrow m.tamano$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---



---

**iOcupar**(in/out  $m$ : mapa, in  $c$ : tupla(int, int))

1:  $m[\Pi_1(c)][\Pi_2(c)] \leftarrow true$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

Justificación: El acceso a una posición de un vector y su modificación es  $\Theta(1)$

---



---



---

**iLibre**(in  $m$ : mapa, in  $c$ : tupla(int, int))  $\rightarrow res$ : bool

1:  $res \leftarrow \neg m[\Pi_1(c)][\Pi_2(c)]$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

Justificación: El acceso a una posición de un vector es  $\Theta(1)$

---



---



---

**iNuevoMapa**(in  $n$ : nat)  $\rightarrow res$ : mapa

1:  $m.tamano \leftarrow n$

$\triangleright \Theta(1)$

2:  $v \leftarrow Vacía()$

$\triangleright \Theta(1)$

3:  $i \leftarrow 0$

$\triangleright \Theta(1)$

4: **while**  $i < n$  **do**

$\triangleright O(n)$

5:      $v.AgregarAtras(false)$

6:      $i \leftarrow i + 1$

7: **end while**

8:  $i \leftarrow 0$

9: **while**  $i < n$  **do**

$\triangleright O(n^2)$

10:      $res.AgregarAtras(v.Copiar())$

$\triangleright O(n)$

11:      $i \leftarrow i + 1$

$\triangleright O(1)$

12: **end while**

Complejidad:  $\Theta(n^2)$

Justificación: Copiar un vector de  $n$  booleanos es  $O(n * copy(bool))$  y copiar un bool es  $\Theta(1)$ . Luego, agregar  $n$  veces la copia del vector es  $O(n^2)$

---

## 4. Módulo Dirección

Aquí va la descripción

### Interfaz

generos: dir.

se explica con: DIRECCIÓN.

### Operaciones básicas de Dirección

ARRIBA()  $\rightarrow res : dir$

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

**Post**  $\equiv \{res =_{obs} \uparrow\}$

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

**Descripción:** genera la dirección arriba.

ABAJO()  $\rightarrow res : dir$

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

**Post**  $\equiv \{res =_{obs} \downarrow\}$

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

**Descripción:** genera la dirección abajo.

IZQUIERDA()  $\rightarrow res : dir$

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

**Post**  $\equiv \{res =_{obs} \leftarrow\}$

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

**Descripción:** genera la dirección izquierda.

DERECHA()  $\rightarrow res : dir$

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

**Post**  $\equiv \{res =_{obs} \rightarrow\}$

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

**Descripción:** genera la dirección derecha.

INVERTIR(in/out  $d : dir$ )

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

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

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

**Descripción:** invierte la dirección.

## Representación

### Representación de Dirección

dir se representa con string

$Rep : dir \rightarrow bool$

$Rep(d) \equiv true \iff$

$d =_{obs} "arriba" \vee$

$d =_{obs} "abajo" \vee$

$d =_{obs} "izquierda" \vee$

$d =_{obs} "derecha"$

$Abs : dir \rightarrow dir$

$Abs(d) =_{obs} d_{tad} : dir \mid (d =_{obs} "arriba" \wedge d_{tad} =_{obs} \uparrow) \vee$   
 $(d =_{obs} "abajo" \wedge d_{tad} =_{obs} \downarrow) \vee$   
 $(d =_{obs} "izquierda" \wedge d_{tad} =_{obs} \leftarrow) \vee$   
 $(d =_{obs} "derecha" \wedge d_{tad} =_{obs} \rightarrow)$

$\{Rep(d)\}$

## Algoritmos

### Algoritmos del módulo

---

---

**iArriba()**  $\rightarrow res : \text{dir}$ 1:  $res \leftarrow \text{"arriba"}$  $\triangleright \Theta(1)$ Complejidad:  $\Theta(1)$ 

---

---

---

**iAbajo()**  $\rightarrow res : \text{dir}$ 1:  $res \leftarrow \text{"abajo"}$  $\triangleright \Theta(1)$ Complejidad:  $\Theta(1)$ 

---

---

---

**iIzquierda()**  $\rightarrow res : \text{dir}$ 1:  $res \leftarrow \text{"izquierda"}$  $\triangleright \Theta(1)$ Complejidad:  $\Theta(1)$ 

---

---

---

**iDerecha()**  $\rightarrow res : \text{dir}$ 1:  $res \leftarrow \text{"derecha"}$  $\triangleright \Theta(1)$ Complejidad:  $\Theta(1)$ 

---

---

---

**iInvertir(in/out d: dir)**1:  $switch(d)$  $\triangleright \Theta(1)$ 2:  $case \text{"arriba"} :$ 3:      $d \leftarrow \text{"abajo"}$ 4:  $case \text{"abajo"} :$ 5:      $d \leftarrow \text{"arriba"}$ 6:  $case \text{"izquierda"} :$ 7:      $d \leftarrow \text{"derecha"}$ 8:  $case \text{"derecha"} :$ 9:      $d \leftarrow \text{"izquierda"}$ Complejidad:  $\Theta(1)$ 

---

## 5. Módulo Acción

Aquí va la descripción

### Interfaz

**generos:** `accion`.

**se explica con:** `ACCIÓN`.

### Operaciones básicas de Acción

`MOVER(in d: dir) → res : accion`

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

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

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

**Descripción:** genera una acción de mover en la dirección especificada.

`PASAR() → res : accion`

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

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

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

**Descripción:** genera la acción de pasar.

`DISPARAR() → res : accion`

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

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

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

**Descripción:** genera la acción de disparar.

`APLICAR() → res : tupla()`

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

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

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

**Descripción:** genera la acción de disparar.

## Representación

### Representación de Acción

El objetivo de este módulo es implementar una lista doblemente enlazada con punteros al principio y al fin. Para simplificar un poco el manejo de la estructura, vamos a reemplazarla por una lista circular, donde el siguiente del último apunta al primero y el anterior del primero apunta al último. La estructura de representación, su invariante de representación y su función de abstracción son las siguientes.

**mapa se representa con m**

donde **m** es `tupla(tamano: nat, casilleros: vec(vec(bool)),)`

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

$\text{Rep}(m) \equiv \text{true} \iff \text{La longitud de m.casilleros es igual a tamano} \wedge$

La longitud del vector m.casilleros es igual a la de todo otro vector dentro de el)  $\wedge$

Es conexa

$\text{Abs} : \text{mapa } m \rightarrow \text{hab}$

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

$\text{Abs}(m) =_{\text{obs}} h : \text{hab} \mid m.\text{tamano} =_{\text{obs}} \text{tam}(h) \wedge_L$

$(\forall t: \text{tuple}(\text{nat}, \text{nat}))(0 \leq \Pi_1(t), \Pi_2(t) < m.\text{tamano} - 1 \Rightarrow_L$

$\text{libre}(m, t) =_{\text{obs}} m.\text{casilleros}[\Pi_1(t)][\Pi_2(t)])$

## Algoritmos

En esta sección se hace abuso de notación en los cálculos de álgebra de órdenes presentes en la justificaciones de los algoritmos. La operación de suma “+” denota secuencialización de operaciones con determinado orden de complejidad,

y el símbolo de igualdad “=” denota la pertenencia al orden de complejidad resultante.

## Algoritmos del módulo

---

**iTam**(in  $m : \text{mapa}$ )  $\rightarrow res : \text{nat}$

1:  $res \leftarrow m.tamano$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---