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

Algoritmos y Estructuras de Datos II, DC, UBA.

1$^{er}$ cuatrimestre de 2019

## Índice

# 1. Diccionario Trie ($\alpha$)

El módulo Diccionario Trie provee un diccionario básico montado sobre un trie.

## Interfaz

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

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

**géneros**: `diccTrie`($string, \alpha$).

### Operaciones básicas de diccionario

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

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

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

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

## Representación

### Representación del diccionario

`diccTrie`($string, \alpha$) **se representa con** `dic`

donde `dic` es `tupla`($raiz$: `puntero(nodo)`)

donde `nodo` es `tupla`($significado$: $\alpha$,
$siguientes$: `arreglo(puntero(nodo))[255]` )

Rep : dic $\longrightarrow$ bool
Rep($d$) $\equiv$ true $\iff$ #claves(secuADicc($d$.claves)) = long($d$.claves) $\wedge$ long($d$.claves) = long($d$.significados)

Abs : diccTrie $d$ $\longrightarrow$ diccTrie($string$, $\alpha$) {Rep($d$)}

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

# Algoritmos

---

**iVacía**() $\rightarrow$ $res$ : diccTrie($string$, $\alpha$)

1: // Reserva la memoria y crea un arreglo estático de 255 posiciones del constructor por defecto de $\alpha$
2: $res \leftarrow \langle raiz : NULL,\ siguientes : arreglo\_estatico[255]\ de\ \alpha \rangle$ ▷ $\Theta(1)$

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

---

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

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

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

---

**iDefinido?**(**in**/**out** $d$: `diccTrie(`$string$`, `$\alpha$`)`, **in** $k$: $string$) $\rightarrow$ $res$ : bool

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

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 ejecutaran dichas asignaciones |k| veces como máximo. Luego la complejidad será $\mathcal{O}(|k|)$.

---

# 2. Módulo Juego

Aqui va la descripción

# Interfaz

**generos**: `juego`. **se explica con**: JUEGO.

## Operaciones básicas de Juego

NUEVOJUEGO(**in** $m$ : `mapa`, **in** $pjs$ : `conj(string)`, **in** $eventosFan$ : `vector(evento)`) $\rightarrow$ $res$ : `juego`
**Pre** $\equiv$ $\{\neg vacio(pjs) \wedge (\forall e : evento)(e \in eventosFan \Rightarrow_{\text{L}} e.pos \in posiciones(m))\}$
**Post** $\equiv$ $\{res =_{\text{obs}} nuevoJuego(m, pjs, eventosFan)\}$
**Complejidad:** $\Theta(?)$
**Descripción:** crea un nuevo juego con el mapa dado, un conjunto de jugadores, y los eventos de un fantasma.

PASAR(**in** $j$ : `juego`) $\rightarrow$ $res$ : `juego`
**Pre** $\equiv$ $\{\text{true}\}$
**Post** $\equiv$ $\{res =_{\text{obs}} pasar(j)\}$
**Complejidad:** $\Theta(?)$
**Descripción:** actualiza sin acción de algún jugador.

STEP(**in** $j$ : `juego`, **in** $a$ : `accion`, **in** $pj$ : `string`) $\rightarrow$ $res$ : `juego`
**Pre** $\equiv$ $\{pj \in jugadores(j) \wedge_{\text{L}} jugadorVivo(pj, j) \wedge \neg esPasar(a)\}$
**Post** $\equiv$ $\{res =_{\text{obs}} pasar(j)\}$
**Complejidad:** $\Theta(?)$
**Descripción:** actualiza con la acción $a$ del jugador $pj$.

JUGADORESVIVOS(**in** $j$ : `juego`) $\rightarrow$ $res$ : `conj(puntero(infoPJ))`
**Pre** $\equiv$ $\{\text{true}\}$
**Post** $\equiv$ $\{(\forall p : puntero(infoPJ))(p \in res \Rightarrow_{\text{L}}$
$(p \rightarrow id \in jugadores(j)) \wedge_{\text{L}}$
$(p \rightarrow vivo? \wedge jugadorVivo(p \rightarrow id, j)) \wedge$
$((\forall e : evento)(e \in p \rightarrow eventos \Rightarrow_{\text{L}}$
$(e.pos =_{\text{obs}} posJugador(p \rightarrow id, j)) \wedge$
$(e.dir =_{\text{obs}} dirJugador(p \rightarrow id, j))))\}$
**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** $\equiv$ $\{\text{true}\}$
**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** $\equiv$ $\{\text{true}\}$
**Post** $\equiv$ $\{res =_{\text{obs}} fantasmaEspecial(j)\}$
**Complejidad:** $\Theta(1)$
**Descripción:** devuelve el fantasma especial.
**Aliasing:** res es una referencia no modificable.

FANTASMASVIVOSQUEDISPARAN(**in** $j$ : `juego`) $\rightarrow$ $res$ : `conj(infoFan)`
**Pre** $\equiv$ $\{\text{true}\}$
**Post** $\equiv$ $\{fantasmaValido(j, res) \wedge_{\text{L}}$
$((\forall f : infoFan)(f \in res \Rightarrow_{\text{L}} disparando(f.eventos, step(j))))\}$
**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** $\equiv$ {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):
$(\forall f : infoFan)(f \in res \Rightarrow_{\text{L}}$
$(f.eventos \in fantasmas(j)) \wedge_{\text{L}}$
$(fantasmaVivo(f.eventos, j)) \wedge$
$((\forall e : evento)(e \in f.eventos \Rightarrow_{\text{L}}$
$(e.pos =_{\text{obs}} posFantasma(f.eventos, j)) \wedge$
$(e.dir =_{\text{obs}} dirFantasma(f.eventos, j))))$

# Representación

## Representación de Juego

**`juego` se representa con `estr`**

donde `j` es `tupla`(// General
*paso*: `nat`,
*ronda*: `nat`,
*mapa*: `m`,

// Disparos
*mapaDisparos*: `arreglo(arreglo(nat))`,
*disparosUltimoPaso*: `conj(posicion)`,

// Jugadores
*jugadores*: `diccTrie(string, itConjLineal(infoPJ))`,
*infoJugadores*: `conj(infoPJ)`,
*infoActualJugadoresVivos*: `conj(infoActualPJ)`,
*jugadoresVivos*: `conj(itConjLineal(infoPJ))`,

// Fantasmas
*infoFantasmas*: `conj(infoFan)`,
*infoActualFantasmasVivos*: `conj(infoActualFan)`,
*fantasmasVivos*: `conj(itConjLineal(infoFan))`,
*fantasmaEspecial*: `itConjLineal(infoActualFan)` )

donde `infoPJ` es `tupla`(*eventos*: `vector(evento)`,
*infoActual*: `itConjLineal(infoActualPJ)` )

donde `infoActualPJ` es `tupla`(*identidad*: `string`,
*posicion*: `pos`,
*direccion*: `dir` )

donde `infoFan` es `tupla`(*eventos*: `vector(evento)` )

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

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

Abs : mapa $m$ $\longrightarrow$ hab {Rep($m$)}
Abs($m$) $=_{\text{obs}}$ 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

---

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

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

Complejidad: $\Theta(1)$

---

# 3. Módulo Mapa

Aqui va la descripción

# Interfaz

**generos**: `mapa`.

**se explica con**: HABITACIÓN.

## Operaciones básicas del mapa

NUEVOMAPA(**in** $n$ : `nat`) $\rightarrow$ $res$ : `mapa`
**Pre** $\equiv$ {true}
**Post** $\equiv$ $\{res =_{\text{obs}} nuevaHab(n)\}$
**Complejidad:** $\Theta(n^2)$
**Descripción:** genera un mapa de tamano n x n.

OCUPAR(**in/out** $m$ : `mapa`, **in** $c$ : $tupla(int, int)$)
**Pre** $\equiv$ $\{m =_{\text{obs}} m_0 \wedge c \in casilleros(m) \wedge_{\text{L}} libre(m, c) \wedge alcanzan(libres(m) - c, libres(m) - c, m)\}$
**Post** $\equiv$ $\{m =_{\text{obs}} ocupar(c, m_0)\}$
**Complejidad:** $\Theta(1)$
**Descripción:** ocupa una posicion del mapa siempre y cuando este no deje de ser conexo.

TAM(**in** $m$ : `mapa`) $\rightarrow$ $res$ : `nat`
**Pre** $\equiv$ {true}
**Post** $\equiv$ $\{res =_{\text{obs}} tam(m)\}$
**Complejidad:** $\Theta(1)$
**Descripción:** devuelve el tamano del mapa.

LIBRE(**in** $m$ : `mapa`, **in** $c$ : $tupla(int, int)$) $\rightarrow$ $res$ : `bool`
**Pre** $\equiv$ $\{c \in casilleros(m)\}$
**Post** $\equiv$ $\{res =_{\text{obs}} libre(c, m)\}$
**Complejidad:** $\Theta(1)$
**Descripción:** devuelve si un elemento esta 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 `tupla`(*tamano*: `nat`,*casilleros*: `vec(vec(bool))`,)

Rep : mapa $\longrightarrow$ bool
Rep($m$) $\equiv$ true $\iff$ 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
Abs : mapa $m$ $\longrightarrow$ hab {Rep($m$)}
Abs($m$) $=_{\text{obs}}$ h: hab | $m.tamano =_{\text{obs}} tam(h) \wedge_{\text{L}}$
($\forall t$: tuple(nat,nat))($0 \leq \Pi_1(t), \Pi_2(t) < m.tamano - 1 \Rightarrow_{\text{L}}$
libre(m, t) $=_{\text{obs}}$ $m.casilleros[\Pi_1(t)][\Pi_2(t)]$)

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

Complejidad: $\Theta(1)$

---

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

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

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

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

---

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

1: $m.tamano \leftarrow n$ ▷ $\Theta(1)$
2: $v \leftarrow Vacia()$ ▷ $\Theta(1)$
3: $i \leftarrow 0$ ▷ $\Theta(1)$
4: **while** $i < n$ **do** ▷ $O(n)$
5: $v.AgregarAtras(false)$
6: $i \leftarrow i + 1$
7: **end while**
8: $i \leftarrow 0$
9: **while** $i < n$ **do** ▷ $O(n^2)$
10: $res.AgregarAtras(v.Copiar())$ ▷ $O(n)$
11: $i \leftarrow i + 1$ ▷ $O(1)$
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

Aqui va la descripción

# Interfaz

**generos**: `dir`.

**se explica con**: DIRECCIÓN.

## Operaciones básicas de Dirección

ARRIBA() $\rightarrow$ *res* : `dir`
**Pre** $\equiv$ {true}
**Post** $\equiv$ {*res* $=_{\text{obs}}\uparrow$}
**Complejidad:** $\Theta(1)$
**Descripción:** genera la dirección arriba.

ABAJO() $\rightarrow$ *res* : `dir`
**Pre** $\equiv$ {true}
**Post** $\equiv$ {*res* $=_{\text{obs}}\downarrow$}
**Complejidad:** $\Theta(1)$
**Descripción:** genera la dirección abajo.

IZQUIERDA() $\rightarrow$ *res* : `dir`
**Pre** $\equiv$ {true}
**Post** $\equiv$ {*res* $=_{\text{obs}}\leftarrow$}
**Complejidad:** $\Theta(1)$
**Descripción:** genera la dirección izquierda.

DERECHA() $\rightarrow$ *res* : `dir`
**Pre** $\equiv$ {true}
**Post** $\equiv$ {*res* $=_{\text{obs}}\rightarrow$}
**Complejidad:** $\Theta(1)$
**Descripción:** genera la dirección derecha.

INVERTIR(**in**/**out** $d$: `dir`)
**Pre** $\equiv$ {true}
**Post** $\equiv$ {*res* $=_{\text{obs}}$ *invertir*($d$)}
**Complejidad:** $\Theta(1)$
**Descripción:** invierte la dirección.

# Representación

## Representación de Dirección

**`dir` se representa con `string`**

Rep : dir $\longrightarrow$ bool
Rep($d$) $\equiv$ true $\iff$
d $=_{\text{obs}}$ "*arriba*" $\vee$
d $=_{\text{obs}}$ "*abajo*" $\vee$
d $=_{\text{obs}}$ "*izquierda*" $\vee$
d $=_{\text{obs}}$ "*derecha*"

Abs : dir $d$ $\longrightarrow$ dir {Rep($d$)}
Abs($d$) $=_{\text{obs}}$ $\text{d}_{tad}$: dir | (d $=_{\text{obs}}$ "*arriba*" $\wedge$ $\text{d}_{tad}$ $=_{\text{obs}}\uparrow$) $\vee$
(d $=_{\text{obs}}$ "*abajo*" $\wedge$ $\text{d}_{tad}$ $=_{\text{obs}}\uparrow$) $\vee$
(d $=_{\text{obs}}$ "*izquierda*" $\wedge$ $\text{d}_{tad}$ $=_{\text{obs}}\leftarrow$) $\vee$
(d $=_{\text{obs}}$ "*derecha*" $\wedge$ $\text{d}_{tad}$ $=_{\text{obs}}\rightarrow$)

# Algoritmos

## Algoritmos del módulo

---

**iArriba**() → *res* : dir

1: *res* ← "*arriba*" ▷ Θ(1)

Complejidad: Θ(1)

---

**iAbajo**() → *res* : dir

1: *res* ← "*abajo*" ▷ Θ(1)

Complejidad: Θ(1)

---

**iIzquierda**() → *res* : dir

1: *res* ← "*izquierda*" ▷ Θ(1)

Complejidad: Θ(1)

---

**iDerecha**() → *res* : dir

1: *res* ← "*derecha*" ▷ Θ(1)

Complejidad: Θ(1)

---

**iInvertir**(**in**/**out** *d* : `dir`)

1: *switch*(*d*) ▷ Θ(1)
2: *case* "*arriba*" :
3: $\quad$ *d* ← "*abajo*"
4: *case* "*abajo*" :
5: $\quad$ *d* ← "*arriba*"
6: *case* "*izquierda*" :
7: $\quad$ *d* ← "*derecha*"
8: *case* "*derecha*" :
9: $\quad$ *d* ← "*izquierda*"

Complejidad: Θ(1)

---

# 5. Módulo Acción

Aqui va la descripción

# Interfaz

**generos**: `accion`.

**se explica con**: ACCIÓN.

## Operaciones básicas de Acción

MOVER(**in** $d$: `dir`) $\rightarrow$ $res$ : `accion`
**Pre** $\equiv$ {true}
**Post** $\equiv$ {$res =_{\text{obs}} mover(d)$}
**Complejidad:** $\Theta(1)$
**Descripción:** genera una acción de mover en la dirección especificada.

PASAR() $\rightarrow$ $res$ : `accion`
**Pre** $\equiv$ {true}
**Post** $\equiv$ {$res =_{\text{obs}} pasar$}
**Complejidad:** $\Theta(1)$
**Descripción:** genera la acción de pasar.

DISPARAR() $\rightarrow$ $res$ : `accion`
**Pre** $\equiv$ {true}
**Post** $\equiv$ {$res =_{\text{obs}} disparar$}
**Complejidad:** $\Theta(1)$
**Descripción:** genera la acción de disparar.

APLICAR() $\rightarrow$ $res$ : `tupla()`
**Pre** $\equiv$ {true}
**Post** $\equiv$ {$res =_{\text{obs}} disparar$}
**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))`,)

Rep : mapa $\longrightarrow$ bool
Rep($m$) $\equiv$ true $\iff$ 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
Abs : mapa $m$ $\longrightarrow$ hab {Rep($m$)}
Abs($m$) $=_{\text{obs}}$ h: hab | $m.tamano =_{\text{obs}} tam(h) \wedge_{\text{L}}$
$(\forall t$: tuple(nat,nat))$(0 \leq \Pi_1(t), \Pi_2(t) < m.tamano - 1 \Rightarrow_{\text{L}}$
libre(m, t) $=_{\text{obs}} m.casilleros[\Pi_1(t)][\Pi_2(t)])$

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

Complejidad: $\Theta(1)$