DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Práctico II

## Algo2Landia - Diseño

Algoritmos y estructuras de datos II

Segundo Cuatrimestre de 2021


| Integrante | LU | Correo electrónico |
|---|---|---|
| Cagnoni, Sebastián | 120/19 | sebacagnoni@gmail.com |
| González, Gerónimo | 34/20 | geronimogonzalez95@gmail.com |
| Ruberto, Stéfano Miyel | 763/19 | stefanomruberto@gmail.com |
| Salguero, Mariano | 716/07 | marianosalguero88@gmail.com |





**Facultad de Ciencias Exactas y Naturales**

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

# Índice

# 1. Módulo Simulación

## Interfaz

**se explica con**: SIMULACIÓN.

**géneros**: `sim`.

**Operaciones básicas de simulación**

CREARSIMULACIÓN(in m : mapa, in i : posición, in obs : dicc(color, pos) ) $\rightarrow res$ : `sim`
**Pre** $\equiv$ { enRango(i, m) $\wedge_L$ (($\forall c$ : color)(def?(c, objetos) $\rightarrow_L$ (enRango(m, obtener(c, objetos)) $\wedge$ obtener(c, objetos) $\neq$ i))}
**Post** $\equiv$ { res $=_{obs}$ nuevaSimulación(m, i, obs)}
**Complejidad:** $\mathcal{O}(a^2 * l + a * l * |C|)$
**Descripción:** Genera una nueva simulación con un agente en la posición i y una serie de objetos con sus respectivos colores y posiciones sobre el mapa (dados por el diccionario obs).
**Aliasing:** No se produce aliasing.

MOVERAGENTE(in/out s : sim, in d : Tupla$\langle Int, Int\rangle$) $\rightarrow res$ : `bool`
**Pre** $\equiv$ { s = $s_0 \wedge$ d $\in \{\langle 1,0\rangle, \langle 0,1\rangle, \langle -1,0\rangle, \langle 0,-1\rangle\}$}
**Post** $\equiv$ { res = hayMovimiento(posJugador($s_0$), d, mapa($s_0$)) $\wedge$ (res $\rightarrow$ s $=_{obs}$ mover($s_0$, d)) $\wedge$ ($\neg$ res $\rightarrow$ s $=_{obs}$ $s_0$}
**Complejidad:** $\mathcal{O}(|C|)$
**Descripción:** De ser posible se modifica el estado de la simulación actualizando la posición del agente, la cantidad de pasos realizados, el conjunto de objetivos realizados (en caso de haberse cumplido alguno) y la posición de los objetos en el mapa (en caso de haberse movido alguno).
**Aliasing:** No se produce aliasing, pero se modifica colateralmente la instancia de la simulación.

AGREGAROBJETIVO(in/out s : sim, in o : objetivo)
**Pre** $\equiv$ { $s = s_0 \wedge$ (colorObjeto(o) $\in$ coloresObjetos($s_0$) $\wedge_L$ def?(colorDestino(o), receptaculos(mapa($s_0$))
}
**Post** $\equiv$ { s = agObjetivo($s_0$, o) }
**Complejidad:** $\mathcal{O}(|C|)$
**Descripción:** Agrega el objetivo pasado por parámetro al conjunto de objetivos disponibles en caso de existir un objeto y un receptáculo de los colores respectivos indicados.
**Aliasing:** No se produce aliasing.

MAPASIMULACION(in s : sim) $\rightarrow res$ : `mapa`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = mapa(s) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Devuelve el mapa de la simulación.
**Aliasing:** Devuelve una referencia no modificable al mapa de la simulacion

POSICIÓNJUGADOR(in s : sim) $\rightarrow res$ : `posición`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = posJugador(s) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Devuelve la posición actual del jugador de la simulación.
**Aliasing:** Devuelve una referencia no modificable a la posición del agente en el mapa.

CANTIDADDEMOVIMIENTOS(in s : sim) $\rightarrow res$ : `Nat`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = cantMovimientos(s) }

**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Devuelve la cantidad de movimientos válidos que realizó el agente.
**Aliasing:** Devuelve una referencia no modificable a la cantidad de movimientos del agente en el mapa.

OBJETIVOSDISPONIBLES(in s : sim) $\rightarrow res$ : `conj(Objetivo)`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = objetivosDisponibles(s) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Devuelve la cantidad de objetivos disponibles en la simulación.
**Aliasing:** Se devuelve una referencia no modificable al conjunto de objetivos disponibles.

CANTIDADOBJETIVOSREALIZADOS(in s : sim) $\rightarrow res$ : `Nat`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = #objetivosRealizados(s) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Devuelve la cantidad de objetivos realizados.
**Aliasing:** La operacion devuelve una referencia no modificable a la cantidad de objetivos realizados.

COLORESDEOBJETOS(in s : sim) $\rightarrow res$ : `conj(Color)`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = coloresObjetos(s) }
**Complejidad:** $\mathcal{O}(n * |C|)$
**Descripción:** Mediante esta operación se obtiene el conjunto de colores de los objetos que se encuentran en el mapa.
**Aliasing:** El conjunto de colores se retorna por copia. No se produce aliasing.

POSICIONOBJETO(in s : sim,in color: c) $\rightarrow res$ : `Pos`
**Pre** $\equiv$ {c $\in$ coloresObjetos(s)}
**Post** $\equiv$ { res = posObjeto(s)}
**Complejidad:** $\mathcal{O}(|C|)$
**Descripción:** Devuelve la posción de un objeto que pertenece a la simulación.
**Aliasing:** La operación retorna una copia de la posición del objeto.

OBJETIVOSREALIZADOS(in s : sim) $\rightarrow res$ : `lista(objetivo)`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { $(\forall o$ : objetivo)(realizado?(o, s) $\rightarrow$ #apariciones(res, o) $\geq$ 1) $\wedge$ longitud(res) = #objetivosRealizados(s) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Devuelve la lista de objetivos realizados durante la simulación.
**Aliasing:** La operación retorna una referencia no modificable a la lista.

**Otras operaciones del módulo**

ESPOSIBLEMOVER(in s : sim, in p : pos, in d : Tupla$\langle Int, Int\rangle$) $\rightarrow res$ : `bool`
**Pre** $\equiv$ { d $\in \{\langle 1,0\rangle, \langle 0,1\rangle, \langle -1,0\rangle, \langle 0,-1\rangle\}$ }
**Post** $\equiv$ { res = hayMovimiento(p, d, mapa(s))}
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Determina si es posible moverse en dirección d desde una posición i en el mapa
**Aliasing:** No se produce aliasing.

SIGUIENTEPOSICIÓN(in p : pos, in d : pos) $\rightarrow res$ : `pos`
**Pre** $\equiv$ { d $\in \{\langle 1,0\rangle, \langle 0,1\rangle, \langle -1,0\rangle, \langle 0,-1\rangle\}$ }
**Post** $\equiv$ { res = siguientePosición(p, d) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Dada una posición, devuelve la siguiente en la dirección d

**Aliasing:** No se produce aliasing.

HAYOBJETOENPOSICIÓN(in p : pos, in s : sim) $\rightarrow res$ : `bool`
**Pre** $\equiv$ { enRango(p, mapa(s)) }
**Post** $\equiv$ { res = hayObjeto(p, s) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Devuelve True si hay un objeto en la posición pasada por parámetro, False en caso contrario.
**Aliasing:** No se produce aliasing.

CREARMAPAOBJETOS(in a : nat, in l : nat, in obs : dicc(color, pos)) $\rightarrow res$ : `vector(vector(celdas))`
**Pre** $\equiv$ {$(\forall c$ : color)(def?(c, obs) $\rightarrow_L$ (0 $\leq \pi_1$(obtener(c, obs)) < a $\wedge$ 0 $\leq \pi_2$(obtener(c, obs)) < l)) $\wedge$ (($\forall c_1, c_2$ : color)(def?($c_1$, obs) $\wedge$ def?($c_2$, obs) $\wedge c_1 \neq c_2$) $\rightarrow$ obtener($c_1$, obs) $\neq$ obtener($c_2$, obs))}
**Post** $\equiv$ {tam(res) = a $\wedge$ ($\forall i$ : nat)(0 $\leq$ i < tam(res) $\rightarrow_L$ tam(res[i]) = l) $\wedge$ ($\forall c$ : $color$)(def?(c, obs) $\rightarrow_L$ res[$\pi_0(obtener(obs, c))$][$\pi_1(obtener(obs, c))$] = $< True, c >$) $\wedge$ ($\forall i, j$ : nat)(i < a $\wedge$ j < l $\wedge_L$ ($\forall c$ : color)(c $\in$ claves(obs, c) $\rightarrow_L$ obtener(obs, c) $\neq \langle i, j \rangle$) $\rightarrow_L$ res[i][j] = $\langle False, \$ \rangle$) }
**Complejidad:** $\mathcal{O}(a^2 * l)$
**Descripción:** Devuelve una matriz de dimensiones a x l. Esta matriz contiene los objetos de la simulación en las posiciones determinadas por el diccionario pasado como parámetro.
**Aliasing:** No se produce aliasing.

# Representación

La simulación está representada por una tupla que contiene a las siguientes estructuras:

- El mapa mapaCasillas contiene toda la información estática de la simulación (indica para cada posición si es una rampa, pared o piso y las casillas especiales).
- La matriz mapaObjetos contiene en cada una de sus posiciones una tupla$< bool, color >$. Esta tupla determina si en la posición dada hay un objeto y en ese caso, su color.
- El trie posicionObjetos dado un color de objeto válido, permite saber su posición sobre el mapa en $\mathcal{O}(|C|)$.
- El trie objDisponibles contiene la información referente a los objetivos disponibles en la simulación. Las claves de este trie serán los colores de los objetos de los objetivos, y a su vez cada clave tiene como significado un conjunto (que también sera implementado con un Trie) cuyos elementos son los receptáculos a los que deberían moverse para cumplirse el objetivo. Esto es así porque puede haber más de un objetivo asociado a un objeto dado. De esta manera concluimos que la complejidad de recorrer todo el trie principal y el secundario es $\mathcal{O}(2 * |C|) = \mathcal{O}(|C|)$
- posiciónActual indica la posición del agente en la simulación. Tenerlo como un atributo separado en la estructura nos permite obtenerlo en $\mathcal{O}(1)$ (lo mismo aplica para cantMovimientos y #objRealizados)
- cantMovimientos indica la cantidad de movimientos que hizo el agente en el transcurso de la simulación.
- #objRealizados indica la cantidad de objetivos realizados en la simulación.

**Representación**

`Simulación` **se representa con** `sim`

donde `sim` es `tupla`(*mapaCasillas*: `mapa`
, *mapaObjetos*: `vector(vector(celdas))`, *posiciónObjetos*: `dicTrie(color, posición)`
, *objDisponibles*: `dicTrie(color, ConjTrie(color))`
, *#objRealizados*: `nat` , *cantMovimientos*: `nat` , *posiciónActual*: `pos` )

donde `celdas` es `tupla`(*hayObjeto*: `bool`, *colorObjeto*: `color`)

**Invariante de representación**

Rep : sim $\longrightarrow$ bool

Rep($s$) $\equiv$ true $\Longleftrightarrow$

alto(s.mapaCasillas) = tam(s.mapaObjetos) $\wedge$ ($\forall i$ : nat)(0 $\leq$ i $<$ tam(s.mapaObjetos) $\rightarrow_L$ tam(s.mapaObjetos[i]) = ancho(s.mapCasillas))

$\wedge$

enRango(s.mapaCasillas , s.pos)

$\wedge_L$

(hayObjeto(s.pos, s) = False)

$\wedge$

($\forall$ $o$ : color)( def?(o, s.posiciónObjetos) $\rightarrow_L$ enRango(s.mapaCasillas, obtener(o, s.posiciónObjetos)) $\wedge_L$ s.mapaObjetos[obtener(o,s.posiciónObjetos)$_0$][obtener(o,s.posiciónObjetos)$_1$] = (True, o))

$\wedge$

($\forall i, j$ : nat)(i $<$ tam(s.mapaObjetos) $\wedge$ j $<$ tam(s.mapaObjetos[0]) $\wedge_L$ ($\forall c$ : color)(c $\in$ claves(s.posiciónObjetos, c) $\rightarrow_L$ obtener(s.posiciónObjetos, c) $\neq \langle i, j \rangle$) $\rightarrow_L$ s.mapaObjetos[i][j] = $\langle False, \$ \rangle$)

$\wedge$

($\forall$ p : pos)(enRango(s.mapaCasillas,p) $\wedge_L$ s.mapaObjetos[p$_0$][p$_1$]$_0$ = true $\rightarrow_L$ def?(s.mapaObjetos[p$_0$][p$_1$]$_1$, s.posicionesObjetos) $\wedge_L$ obtener(s.mapaObjetos[p$_0$][p$_1$]$_1$, s.posicionesObjetos) = p)

$\wedge$

($\forall$ $o$ : color)(o $\in$ claves(s.objetivosDisponibles) $\rightarrow_L$ def?(o,s.posiciónObjetos) $\wedge$ ($\forall$ $r$ : color)(r $\in$ obtener(o,s.objetivosDisponibles) $\rightarrow_L$ def?(r,receptáculos(s.mapaCasillas)))

$\wedge$

#objRealizados = #objetivosRealizados

**Función de abstracción**

Abs : sim $e$ $\longrightarrow$ simulación {Rep($e$)}

Abs($e$) $=_{\text{obs}}$ s: simulación | mapa(s) = e.mapaCasilla $\wedge$ posiciónJugador(s) = e.posiciónActual $\wedge$ cantMovimientos(s) = e.cantMovimientos $\wedge$ #objetivosRealizados(s) = e.#objRealizados $\wedge$ coloresObjetos(s) = claves(e.posicionObjetos) $\wedge_L$ ($\forall$ c : color)(c $\in$ coloresObjetos(s) $\rightarrow_L$ posObjeto(s, c) = obtener(e.posiciónObjetos, c)) $\wedge_L$ ($\forall o$: objetivo)(o $\in$ objetivosDisponibles(s) $\leftrightarrow$ colorObjeto(o) $\in$ claves(s.objDisponibles) $\wedge_L$ colorDestino(o) $\in$ claves(obtener(s. objDisponibles, colorObjeto(o))))

# Algoritmos

**Algoritmos del módulo**

**iCrearSimulación**(**in** $m$ : `mapa`, **in** $i$ : `pos`, **in** $obs$ : `dicc(color, pos)`) $\rightarrow$ $res$ : sim

1: objetos $\leftarrow$ crearMapaObjetos(obs) ▷ $\mathcal{O}(a^2 * l)$
2: objetivosDisponibles $\leftarrow$ crearTrie() ▷ $\mathcal{O}(1)$
3: posicionObjs $\leftarrow$ crearTrie() ▷ $\mathcal{O}(1)$
4: itDicc $\leftarrow$ crearIt(obs) ▷ $\mathcal{O}(1)$
5: **while** haySiguiente(itDicc) **do** ▷ $\mathcal{O}$(a * l)
6: definir(posicionObjs, $siguiente(itDicc)_0$, $siguiente(itDicc)_1$) ▷ $\mathcal{O}(|C|)$
7: avanzar(itDicc) ▷ $\mathcal{O}(1)$
8: **end while**
9: $res \leftarrow \langle m, objetos, posicionObjs, objetivosDisponibles, objetivosRealizados, 0, 0, i \rangle$ ▷ $\mathcal{O}(a * l(a + |C|))$

Complejidad: $\mathcal{O}(a^2 * l + a * l * |C|)$

**iMoverAgente**(**in/out** $s$ : sim, **in** $d$ : Tupla$\langle Int, Int\rangle$) $\rightarrow res$ : bool

```
1: if estáEnRango(s.mapaCasillas, siguientePosición(posiciónJugador(s), d)) then                ▷ O(1)
2:     s.cantMovimientos ← s.cantidadMovimientos + 1                                            ▷ O(1)
3: end if
4: mapaObjetos ← s.mapaObjetos                                                                  ▷ O(1)
5: mapaCasillas ← s.mapaCasillas                                                                ▷ O(1)
6: posSiguiente ← siguientePosición(posiciónJugador(s), d)                                      ▷ O(1)
7: posSiguienteSig ← siguientePosición(posSiguiente, d)                                         ▷ O(1)
8: chequeo ← estáEnRango(mapaCasillas, posSiguiente) ∧_L
9: ((!hayObjetoEnPosicion(posSiguiente, s) ∨
10: (esPosibleMover(s, posSiguiente, d) ∧_L !(hayObjetoEnPosicion(posSiguienteSig, s)))))       ▷ O(1)
11: res ← esPosibleMover(s, posicionJugador(s), d) ∧ chequeo                                    ▷ O(1)
12: if res then                                                                                 ▷ O(|C|)
13:     if iHayObjetoEnPosicion(posSiguiente, s) then                                           ▷ O(|C|)
14:         mapaObjetos[posSiguiente].hayObjeto ← false                                         ▷ O(1)
15:         mapaObjetos[posSiguienteSig].hayObjeto ← true                                       ▷ O(1)
16:         mapaObjetos[posSiguienteSig].colorObjeto ← mapaObjetos[posSiguiente].colorObjeto    ▷
    O(1)
17:         mapaObjetos[posSiguiente].colorObjeto ← "$"                                         ▷ O(1)
18:         colorObjetoMovido ← mapaObjetos[posSiguienteSig].colorObjeto                        ▷ O(1)
19:         definir(s.posicionObjetos, colorObjetoMovido , posSiguienteSig)                     ▷ O(|C|)
20:         cumpliUnObjetivo ← false                                                            ▷ O(1)
21:         if mapaCasillas[posSiguienteSig].receptaculo then                                   ▷ O(|C|)
22:             colorReceptaculo ← mapaCasillas[posSiguienteSig].colorReceptaculo               ▷ O(1)
23:             definido ← estaDefinido?(s.objDisponibles, colorObjetoMovido)                   ▷ O(1)
24:             if definido then                                                                ▷ O(|C|)
25:                 trieReceptaculos ← obtenerSignificado(s.objDisponibles, colorObjetoMovido)  ▷ O(|C|)
26:                 if estaDefinido?(trieReceptaculos, colorReceptaculo) then                   ▷ O(|C|)
27:                     cumpliUnObjetivo ← true                                                 ▷ O(1)
28:                 end if
29:             end if
30:         end if
31:         if cumpliUnObjetivo then                                                            ▷ O(|C|)
32:             if #claves(trieReceptaculos) = 1 then                                           ▷ O(|C|)
33:                 eliminar(trieReceptaculos)                                                  ▷ O(|C|)
34:                 eliminarRama(s.objetosDisponiblesBusqueda, colorObjetoMovido)               ▷ O(|C|)
35:             else
36:                 eliminarRama(trieReceptaculos, colorReceptaculo)                            ▷ O(|C|)
37:             end if
38:             #objRealizados ← #objRealizados + 1                                             ▷ O(1)
39:             objetivoRealizado ← nuevoObjetivo(colorObjetoMovido, colorReceptaculo)          ▷ O(1)
40:         end if
41:     end if
42:     s.posicionActual ← posSiguiente                                                         ▷ O(1)
43: end if
```

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

**iAgregarObjetivo(in/out** $s$ : `sim`, **in** $o$ : `objetivo`)

```
1: if estaDefinido?(s.objetivosDisponibles, colorObjeto(o)) then          ▷ O(|C|)
2:     receptaculos ← obtener(s.objetivosDisponibles, colorObjeto(o))     ▷ O(|C|)
3:     agregar(receptaculos, colorDestino(o))                             ▷ O(|C|)
4: else
5:     receptaculos ← vacio()                                             ▷ O(|C|)
6:     agregar(receptaculos, colorDelDestino(o))                          ▷ O(|C|)
7:     definir(s.objDisponibles, colorDelObjeto(o), receptaculos)         ▷ O(|C|)
8: end if
```

Complejidad: $\mathcal{O}(|C|)$
justificación: $\mathcal{O}(2 * |C|) + \mathcal{O}(3 * |C|) = \mathcal{O}(3 * |C|) = \mathcal{O}(|C|)$

---

**iMapaSimulacion(in** $s$ : `sim`) $\rightarrow$ $res$ : mapa

```
1: res ← s.mapaCasillas                                                   ▷ O(1)
```

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

---

**iPosicionJugador(in** $s$ : `sim`) $\rightarrow$ $res$ : pos

```
1: res ← s.posicionActual                                                 ▷ O(1)
```

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

---

**iCantidadDeMovimientos(in** $s$ : `sim`) $\rightarrow$ $res$ : Nat

```
1: res ← s.cantMovimientos                                                ▷ O(1)
```

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

---

**iObjetivosDisponibles(in** $s$ : `sim`) $\rightarrow$ $res$ : conj(Objetivo)

```
1: res ← s.objDisponibles                                                 ▷ O(1)
```

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

---

**iCantidadObjetivosRealizados(in** $s$ : `sim`) $\rightarrow$ $res$ : Nat

```
1: res ← s.#objRealizados                                                 ▷ O(1)
```

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

---

**iColoresDeObjetos**(**in** $s$ : `sim`) $\rightarrow$ $res$ : conj(Color)

```
1: conjuntoColores ← vacio()                                          ▷ O(1)
2: it ← crearIt(s.posicionObjetos)                                    ▷ O(1)
3: while haySiguiente(it) do                                          ▷ O(#Claves(s.posicionObjetos))
4:     agregarRapido(conjuntoColores, siguienteClave(it))             ▷ O(|C|)
5: end while
6: res ← conjuntoColores                                              ▷ O(1)
```

Complejidad: $\mathcal{O}(n * |C|)$
justificación: n denota la cantidad de claves definidas sobre el arbol de objetos, mientras que $|C|$ denota el color con nombre mas largo de todo el conjunto de claves : $\mathcal{O}(n)*\mathcal{O}(|C|)+\mathcal{O}(1) = \mathcal{O}(n*|C|)$

---

**iPosicionObjeto**(**in** $s$ : `sim`, **in** $c$ : `color`) $\rightarrow$ $res$ : pos

```
1: res ← obtener(s.posicionObjetos, c)                                ▷ O(1)
```

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

---

**iEsPosibleMover**(**in** $s$ : `sim`,**in** $p$ : `pos` ,**in** $d$ : `Tupla`$\langle Int, Int\rangle$) $\rightarrow$ $res$ : bool

```
1: mapa ← mapaSimulacion(s)                                           ▷ O(1)
2: posSiguiente ← siguientePosicion(pos, s)                           ▷ O(1)
3: res ← False                                                        ▷ O(1)
4: if iEnRango(mapa(s), posSiguiente) ∧
5: (!hayUnaPared(mapa(s), posSiguiente) ∨ hayUnaRampa(mapa(s), pos)) then   ▷ O(1)
6:     res ← True                                                     ▷ O(1)
7: end if                                                             ▷ O(1)
```

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

---

**iSiguientePosicion**(**in** $p$ : `pos`, **in** $d$ : `Tupla`$\langle Int, Int\rangle$) $\rightarrow$ $res$ : pos

```
1: res ←< s.posicionActual_0 + d_0, s.posicionActual_1 + d_1 >        ▷ O(1)
```

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

---

**iHayObjetoEnPosicion**(**in** $p$ : `pos`, **in** $s$ : `sim`) $\rightarrow$ $res$ : bool

```
1: res ← s.mapaObjetos[p_0][p_1].hayObjeto                            ▷ O(1)
```

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

---

**iObjetivosRealizados**(**in** $s$ : `sim`) $\rightarrow$ $res$ : lista(objetivo)

```
1: res ← s.objetivosRealizados                                        ▷ O(1)
```

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

**iCrearMapaObjetos**(**in** $a$ : `nat`, **in** $l$ : `nat`, **in** $obs$ : `dicTrie(color, pos)` $\rightarrow$ $res$ :vector(vector(celdas))

```
1:  vectorGrande ← vacia()                                              ▷ O(1)
2:  vectorChico ← vacia()                                               ▷ O(1)
3:  while i < l do                                                      ▷ O(l^2)
4:      agregarAtras(vectorChico,< False, $ >)                          ▷ O(l)
5:      i ← i + 1                                                       ▷ O(1)
6:  end while
7:  while j < a do                                                      ▷ O(a^2 * l)
8:      agregarAtras(vectorGrande,vectorChico)                          ▷ O(a * l)
9:      j ← j + 1
10: end while                                                           ▷ O(1)
11: it ← crearIt(obs)                                                   ▷ O(1)
12: while haySiguiente(it) do                                           ▷ O(a * l)
13:     clave ← siguienteClave(it)                                      ▷ O(1)
14:     sig ← siguienteSignificado(it)                                  ▷ O(1)
15:     vectorGrande[sig_0][sig_1].hayObjeto ← True                     ▷ O(1)
16:     vectorGrande[sig_0][sig_1].colorObjeto ← clave                  ▷ O(1)
17:     Avanzar(it)                                                     ▷ O(1)
18: end while
19: res ← vectorGrande                                                  ▷ O(a^2 * l)
```

Complejidad: $\mathcal{O}(a^2 * l)$

# 2. Módulo Mapa

## Interfaz

**se explica con**: MAPA.

**géneros**: `mapa`.

**Operaciones básicas de mapa**

CREARMAPA(in a : nat, in l : nat, in rs : dicc(color, pos)) $\rightarrow res$ : `mapa`
**Pre** $\equiv$ { $(\forall c$ : color)(def?(c, obs) $\rightarrow_L$ (0 $\leq \pi_1$(obtener(c, rs)) $<$ a $\wedge$ 0 $\leq \pi_2$(obtener(c, rs)) $<$ l)) $\wedge$ $((\forall c_1, c_2$ : color)(def?($c_1$, rs) $\wedge$ def?($c_2$, rs) $\wedge c_1 \neq c_2$) $\rightarrow$ obtener($c_1$, rs) $\neq$ obtener($c_2$, rs)) }
**Post** $\equiv$ {res = nuevoMapa(a, l, rs)}
**Complejidad:** $\mathcal{O}(a^2 * l)$
**Descripción:** Devuelve un mapa de a filas y l columnas, con casillas especiales en las posiciones dadas por el diccionario rs.
**Aliasing:** La operacion devuelve un mapa por copia.

AGREGARPAREDENMAPA(in/out m : mapa, in p : pos)
**Pre** $\equiv$ {$m = m_0 \wedge$ (enRango(m, p) $\wedge_L$ esPiso(m, p) $\wedge$ ((∀p′ : pos)(enRango(m,p') $\wedge_L$ dist(p, p') = 1 $\wedge$ esRampa(m, p')) $\rightarrow_L$ ((∃ p" : pos)p $\neq$ p" $\wedge$ enRango(m, p") $\wedge_L$ dist(p',p") = 1 $\wedge$ esPiso(m, p"))))}
**Post** $\equiv$ {m = agregarPared($m_0$, p))}
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Agrega una pared en la posicion del mapa pasada como parámetro. Si la posicion dada no es valida para insertar una pared, el mapa no se modifica y se devuelve False
**Aliasing:** La operacion no produce aliasing, pero se modifica colateralmente al mapa pasado como parámetro.

AGREGARRAMPAENMAPA(in/out m : mapa, in p : pos) $\rightarrow res$ : `bool`
**Pre** $\equiv$ {$m = m_0 \wedge$ (enRango(m, p) $\wedge_L$ esPiso(m, p) $\wedge$ (((∃p′ : pos)(enRango(m, p') $\wedge_L$ dist(p', p) = 1 $\wedge_L$ esPared(m, p')) $\wedge$
(((∃p′ : pos)(enRango(m, p') $\wedge_L$ dist(p', p) = 1 $\wedge_L$ esRampa(m, p')) $\wedge$
((∀p′ : pos)(enRango(m,p') $\wedge_L$ dist(p, p') = 1 $\wedge$ esRampa(m, p')) $\rightarrow_L$ ((∃ p" : pos)p $\neq$ p" $\wedge$ enRango(m, p") $\wedge_L$ dist(p',p") = 1 $\wedge$ esPiso(m, p")))) }
**Post** $\equiv$ { m = agregarPared($m_0$, p)) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** La operacion no produce aliasing, pero se modifica colateralmente al mapa pasado como parámetro.

DIMENSIONESMAPA(in m : mapa) $\rightarrow res$ : `tupla(Nat, Nat)`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = <alto(m), ancho(m)>}
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Devuelve las dimensiones del mapa
**Aliasing:** No se produce aliasing.

CASILLASESPECIALES(in m : mapa) $\rightarrow res$ : `dicc(color, pos)`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = receptaculos(m) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Devuelve un diccionario con todas las casillas de colores que se encuentran sobre el mapa.
**Aliasing:** La operacion devuelve una referencia no modificable al diccionario que contiene las casillas especiales.

HAYUNAPARED?(in m : mapa, in p : pos) $\rightarrow res$ : `bool`
**Pre** $\equiv$ { enRango(m, p) }
**Post** $\equiv$ { res = hayPared(m, p) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Indica si en la posicion p hay una pared
**Aliasing:** No se produce aliasing.

HAYUNARAMPA?(in m : mapa, in p : pos) $\rightarrow res$ : `bool`
**Pre** $\equiv$ { enRango(m, p) }
**Post** $\equiv$ { res = hayRampa(m, p) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Indica si en la posicion p hay una rampa
**Aliasing:** No se produce aliasing.

ESTAENRANGO?(in m : mapa, in p : pos) $\rightarrow res$ : `bool`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = enRango(m, p) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Determina si la posicion se encuentra entre los limites del mapa
**Aliasing:** No se produce aliasing.

**Otras operaciones del módulo**

DISTANCIA(in p : pos, in q : pos) $\rightarrow res$ : `nat`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = $|p_1 - q_1|$ + $|p_2 - q_2|$ }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Determina la distancia entre dos posiciones
**Aliasing:** No se produce aliasing.

# Representación

La siguiente estructura nos permite concentrar toda la información estática del mapa en dos lugares diferentes. Con la matriz es posible acceder a cada posición del mapa en $\mathcal{O}(1)$ y preguntar qué características tiene la misma. Para ello, elegimos una tupla de tres booleanos y un color, donde los primeros dos indican cuál de las tres variantes de superficie es una celda dada (piso, rampa o pared), el tercero si la celda tiene un color y por último el color correspondiente (en caso de haberlo). En este último atributo habrá un signo "$" cuando la celda no tiene color (representaria un string vacío).

Por otro lado, tenemos un diccionario que guarda la información de las casillas especiales: los colores son las claves, y las posiciones son los significados asociados. Esto nos permite saber la posición de una celda coloreada sin tener que recorrer todo el mapa.

**Representación**

`Mapa` **se representa con** `m`

donde `m` es `tupla`(*map*: `Arr(Arr[datosCelda])`, *ancho*: `Int` , *alto*: `Int` , *casillasEspeciales*: `Dicc(color, pos)`)

donde `datosCelda` es `tupla`(*rampa*: `Bool`, *pared*: `Bool`, *receptaculo*: `Bool`, *colorReceptaculo*: `Color`)

**Invariante de representacion**
Rep : map $\longrightarrow$ bool

Rep($m$) $\equiv$ true $\iff$
tam(m.map) = m.alto $\wedge_L$ ($\forall i$ : nat)(0 $\leq$ i $<$ tam(m.map) $\rightarrow_L$ tam(m.map[i]) = m.ancho) $\wedge_L$

($\forall i, j$ : nat)(0 $\leq$ i $<$ tam(m.map) $\wedge_L$ 0 $\leq$ j $<$ tam(m.map[i]) $\rightarrow_L$
(m.map[i][j].rampa $\rightarrow$ $\neg$ m.map[i][j].pared ) $\vee$ (m.map[i][j].pared $\rightarrow$ $\neg$ m.map[i][j].rampa) $\wedge_L$

($\forall c$ : color)(c $\in$ claves(m.casillasEspeciales) $\rightarrow_L$
(m.map[$obtener(c, m.casillasEspeciales)_0$][$obtener(c, m.casillasEspeciales)_1$].Receptaculo
$\wedge$ m.map[$obtener(c, m.casillasEspeciales)_0$][$obtener(c, m.casillasEspeciales)_1$].colorReceptaculo = c)) $\wedge_L$

($\forall p$ : $pos$)(enRango(m, p) $\wedge_L$ m.map[$p_0$][$p_1$].receptaculo $\rightarrow_L$ def?(m.map[$p_0$][$p_1$].colorReceptaculo, m.casillasEspeciales) $\wedge_L$ obtener(m.map[$p_0$][$p_1$].color, m.casillasEspeciales) = p) $\wedge$ ($\forall p$ : $pos$)(enRango(m, p) $\wedge_L$ $\neg$ m.map[$p_0$][$p_1$].receptaculo $\rightarrow_L$ m.map[$p_0$][$p_1$].colorReceptaculo = "$"))

**Funcion de abstraccion**
Abs : map $e$ $\longrightarrow$ mapa {Rep($e$)}
Abs($e$) $=_{\text{obs}}$ m: mapa | ancho(m) = e.ancho $\wedge_L$ alto(m) = e.alto $\wedge_L$
receptaculos(m) = e.casillasEspeciales $\wedge_L$
($\forall$ p : pos)(enRango(m, p) $\rightarrow_L$ (esPared(m, p) = hayUnaPared?(e, p) $\wedge$ esRampa(m, p) = hayUnaRampa?(e, p)))

# Algoritmos

**icrearMapa(in** $a$ : `nat`, **in** $l$ : `nat`, **in** $rs$ : `dicc(color,pos)`) $\rightarrow$ $res$ : mapa

1: vectorChico $\leftarrow$ vacia() ▷ $\mathcal{O}(1)$
2: j $\leftarrow$ 0 ▷ $\mathcal{O}(1)$
3: **while** $j < l$ **do** ▷ $\mathcal{O}(l^2)$
4: agregarAtras(vectorChico, $< False, False, False, \$ >$) ▷ $\mathcal{O}(l)$
5: $j \leftarrow j + 1$ ▷ $\mathcal{O}(1)$
6: **end while**
7: vectorGrande $\leftarrow$ vacia() ▷ $\mathcal{O}(1)$
8: i $\leftarrow$ 0 ▷ $\mathcal{O}(1)$
9: **while** $i < a$ **do** ▷ $\mathcal{O}(a^2 * l)$
10: agregarAtras(vectorGrande, vectorChico) ▷ $\mathcal{O}(a * l)$
11: $i \leftarrow i + 1$ ▷ $\mathcal{O}(1)$
12: **end while** ▷ $\mathcal{O}(1)$
13: it $\leftarrow$ crearIt(rs) ▷ $\mathcal{O}(1)$
14: **while** haySiguiente(it) **do** ▷ $\mathcal{O}(a * l)$
15: clave $\leftarrow$ siguienteClave(it) ▷ $\mathcal{O}(1)$
16: sig $\leftarrow$ siguienteSignificado(it) ▷ $\mathcal{O}(1)$
17: vectorGrande[$sig_0$][$sig_1$].receptaculo $\leftarrow$ True ▷ $\mathcal{O}(1)$
18: vectorGrande[$sig_0$][$sig_1$].colorReceptaculo $\leftarrow$ clave ▷ $\mathcal{O}(1)$
19: Avanzar(it) ▷ $\mathcal{O}(1)$
20: **end while**
21: $res \leftarrow \langle vectorGrande, a, l, rs \rangle$ ▷ $\mathcal{O}(a^2 * l)$

Complejidad: $\mathcal{O}(l^2) + \mathcal{O}(a^2 * l) + \mathcal{O}(a * l) = \mathcal{O}(a^2 * l)$

**iAgregarPared**(**in/out** $m$ : mapa, **in** $p$ : pos)

1: $m[p_0][p_1].pared \leftarrow True$ ▷ $\mathcal{O}(1)$

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

**iAgregarRampa**(**in/out** $m$ : mapa, **in** $p$ : pos)

1: $m[p_0][p_1].rampa \leftarrow True$ ▷ $\mathcal{O}(1)$

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

**iDimensionesMapa**(**in** $m$ : mapa) $\rightarrow res$ : tupla(Nat,Nat)

1: $res \leftarrow \langle m.alto, m.ancho \rangle$ ▷ $\mathcal{O}(1)$

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

**iCasillasEspeciales**(**in** $m$ : mapa) $\rightarrow res$ : dicc(color, pos)

1: $res \leftarrow m.casillasEspeciales$ ▷ $\mathcal{O}(1)$

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

**iHayUnaPared**(**in** $m$ : mapa, **in** $p$ : pos) $\rightarrow res$ : bool

1: $res \leftarrow False$ ▷ $\mathcal{O}(1)$
2: **if** $m[p_0][p_1]$.pared = 1 **then** ▷ $\mathcal{O}(1)$
3: $res \leftarrow True$ ▷ $\mathcal{O}(1)$
4: **end if**

Complejidad: $\mathcal{O}(1)$
justificación: Los accesos a una posicion en el mapa y a los atributos se realizan en tiempo constante.

**iHayUnaRampa**(**in** $m$ : mapa, **in** $p$ : pos) $\rightarrow res$ : bool

1: $res \leftarrow False$ ▷ $\mathcal{O}(1)$
2: **if** $m[p_0][p_1]$.rampa = 1 **then** ▷ $\mathcal{O}(1)$
3: $res \leftarrow True$ ▷ $\mathcal{O}(1)$
4: **end if**

Complejidad: $\mathcal{O}(1)$
justificación: Los accesos a una posicion en el mapa y a los atributos se realizan en tiempo constante.

**iEstaEnRango**(**in** $m$ : `mapa`, **in** $p$ : `pos`) $\rightarrow$ $res$ : bool

1: $res \leftarrow False$ ▷ $\mathcal{O}(1)$
2: $ancho \leftarrow m.ancho$ ▷ $\mathcal{O}(1)$
3: $largo \leftarrow m.alto$ ▷ $\mathcal{O}(1)$
4: **if** $(0 \leq p_0 \wedge p_0 < ancho) \wedge (0 \leq p_1 \wedge p_1 < largo)$ **then** ▷ $\mathcal{O}(1)$
5: $\quad res \leftarrow True$ ▷ $\mathcal{O}(1)$
6: **end if**

Complejidad: $\mathcal{O}(1)$
justificación: Los accesos a los atributos del mapa se realizan en tiempo constante.

**iDistancia**(**in** $p$ : `pos`, **in** $q$ : `pos`) $\rightarrow$ $res$ : nat

1: $res \leftarrow |p_0 - q_0| + |p_1 - q_2|$ ▷ $\mathcal{O}(1)$

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

# 3. Módulo Objetivo

El módulo Objetivo provee una forma de generar objetivos y acceder a la información relevante.

## Interfaz

**se explica con**: OBJETIVO.

**géneros**: `objetivo`

**Operaciones básicas de objetivo**

NUEVOOBJETIVO(in $c_1$ : color, in $c_2$ : color) $\rightarrow res$ : `objetivo`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = nuevoObjetivo($c_1, c_2$)}
**Complejidad:** $\mathcal{O}(max(copy(c_1), copy(c_2))$
**Descripción:** Crea un objetivo con un primer parámetro objeto y segundo parámetro destino
**Aliasing:** Se devuelve una copia

COLOROBJETO(in o : objetivo) $\rightarrow res$ : `color`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = colorObjeto(o) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Devuelve el color del objeto
**Aliasing:** Se devuelve una referencia no modificable al color

COLORDESTINO(in o : objetivo) $\rightarrow res$ : `color`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = colorDestino(o) }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Se devuelve el color del destino
**Aliasing:** Se devuelve una referencia no modificable al color

## Representación

En este módulo vamos a utilizar dos strings para representar un objetivo genérico.

**Representación**

`Objetivo` **se representa con** `obj`

donde `obj` es `tupla`(*colorObjeto*: `String` , *colorDestino*: `String` )

**Invariante de representacion**
Rep : obj $\longrightarrow$ bool
Rep($o$) $\equiv$ True

**Funcion de abstraccion**
Abs : obj $e$ $\longrightarrow$ objetivo {Rep($e$)}
Abs($e$) $=_{\text{obs}}$ o: objetivo | colorObjeto(o) = e.ColorObjeto $\wedge$ colorDestino(o) = e.ColorDestino

# Algoritmos

**iNuevoObjetivo**(**in** $c$ : `color`, **in** $d$ : `color`) $\rightarrow$ $res$ : objetivo

1: $res \leftarrow \langle c, d \rangle$ ▷ $\mathcal{O}(max(copy(c), copy(d))$

Complejidad: $\mathcal{O}(max(copy(c), copy(d))$
Justificación: El algoritmo toma los dos parámetros por copia. Al momento de devolver el objetivo se debe generar una copia de cada string, por lo que la complejidad queda determinada por: $\mathcal{O}(copy(c)) + \mathcal{O}(copy(d)) = \mathcal{O}(max(copy(c), copy(c))$

**iColorObjeto**(**in** $o$ : `objetivo`) $\rightarrow$ $res$ : color

1: $res \leftarrow o.colorObjeto$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$
Justificación: La complejidad es constante ya que el algoritmo solo implica acceder a un elemento de la estructura.

**iColorDestino**(**in** $o$ : `objetivo`) $\rightarrow$ $res$ : color

1: $res \leftarrow o.colorDestino$ ▷ $\mathcal{O}(1)$

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

# 4. Módulos auxiliares

En esta sección se definen las interfaces de las abstracciones utilizadas en los módulos anteriores.

## 4.1. Módulo Trie

# Interfaz

**parámetros formales**
**géneros** $\alpha$, $\beta$

**géneros**: `dicTrie(`$\alpha$`, `$\beta$`)`

**Operaciones básicas de dicTrie**

CREARTRIE() $\rightarrow res$ : `dicTrie(`$\alpha$`, `$\beta$`)`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = vacio }
**Complejidad:** $\mathcal{O}(1)$
**Descripción:** Crea un trie sin claves definidas.
**Aliasing:** No se produce aliasing.

DEFINIR(in/out d : dicTrie($\alpha, \beta$), in clave : $\alpha$, in significado : $\beta$)
**Pre** $\equiv$ { $d = d_0$ }
**Post** $\equiv$ { d = definir($d_0$, clave, significado) }
**Complejidad:** $\mathcal{O}(|clave|)$
**Descripción:** Agrega un significado nuevo con su respectiva clave al Trie. Si la clave ya estaba definida, modifica su significado.
**Aliasing:** No se produce aliasing. Se modifica colateralmente al diccionario pasado como parámetro.

OBTENERSIGNIFICADO(in d : dicTrie($\alpha$, $\beta$), in clave : $\alpha$) $\rightarrow res$ : $\beta$
**Pre** $\equiv$ { def?(d, clave) }
**Post** $\equiv$ { res = obtener(d, clave, significado)}
**Complejidad:** $\mathcal{O}(|clave|)$
**Descripción:** Retorna el significado asociado a una clave válida del diccionario.
**Aliasing:** El significado se devuelve por referencia

ESTÁDEFINIDO?(in d : dicTrie($\alpha, \beta$), in clave : $\alpha$) $\rightarrow res$ : `bool`
**Pre** $\equiv$ { True }
**Post** $\equiv$ { res = def?(d, clave)}
**Complejidad:** $\mathcal{O}(|clave|)$
**Descripción:** Si la clave está definida en el diccionario se devuelve True, en caso contrario, el valor de retorno será False.
**Aliasing:** No se produce aliasing.