DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Practico N°1

Modulo de Diseño SimCity

19 de junio de 2022 — Algoritmos y Estructura de Datos 2

Grupo 33

Integrante	LU	Correo electrónico
Alvarez Motta, Facundo	889/21	facundoalvarezmotta@gmail.com
Acuña, Martin	596/21	acunamartin1426@gmail.com
Solar, Facundo	493/21	solarfacundo@gmail.com
Zambrana Zamudio, Cristian	871/19	cristian9rk@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
https://exactas.uba.ar

1. Especificación

TAD MAPA

igualdad observacional

$$(\forall m, m' : \text{Mapa}) \left(m =_{\text{obs}} m' \iff \begin{pmatrix} \text{horizontales}(m) =_{\text{obs}} \text{horizontales}(m') \wedge_{\text{L}} \\ \text{verticales}(m) =_{\text{obs}} \text{verticales}(m') \end{pmatrix} \right)$$

géneros Mapa

exporta Mapa, generadores, observadores

usa CONJUNTO(NAT), NAT

observadores básicos

horizontales : Mapa $\longrightarrow$ conj(Nat)
verticales : Mapa $\longrightarrow$ conj(Nat)

generadores

crear : conj(Nat) $\times$ conj(Nat) $\longrightarrow$ Mapa

axiomas $\forall hs, vs$: conj(Nat)

horizontales(crear(hs, vs)) $\equiv$ hs
verticales(crear(hs, vs)) $\equiv$ vs

Fin TAD

CONSTRUCCION es enum{casa, comercio}, NIVEL es Nat, POS es tupla(Nat, Nat)

TAD SIMCITY

igualdad observacional

$$(\forall s, s' : \text{SimCity}) \left(s =_{\text{obs}} s' \iff \begin{pmatrix} \text{mapa}(s) =_{\text{obs}} \text{mapa}(s') \wedge_{\text{L}} \\ \text{casas}(s) =_{\text{obs}} \text{casas}(s') \wedge_{\text{L}} \\ \text{comercios}(s) =_{\text{obs}} \text{comercios}(s') \wedge_{\text{L}} \\ \text{popularidad}(s) =_{\text{obs}} \text{popularidad}(s') \wedge_{\text{L}} \end{pmatrix} \right)$$

géneros SimCity

exporta generadores, observadores, SimCity, otras operaciones

usa POS, NIVEL, DICCIONARIO(POS, NIVEL), NAT, MAPA, CONSTRUCCION

observadores básicos

mapa : SimCity $\longrightarrow$ Mapa
casas : SimCity $\longrightarrow$ dicc(Pos, Nivel)
comercios : SimCity $\longrightarrow$ dicc(Pos, Nivel)
popularidad : SimCity $\longrightarrow$ Nat

generadores

iniciar : Mapa $\longrightarrow$ SimCity

avanzarTurno : SimCity s $\times$ dicc(Pos $\times$ Construccion) cs $\longrightarrow$ SimCity
$\left\{ \begin{array}{l} \#\text{claves(cs)} \geq 1 \wedge_{\text{L}} \ \mathit{OcupPorRios?}(\mathit{claves}(cs), \mathit{mapa}(s)) \wedge_{\text{L}} \\ \mathit{claves}(cs) \cap \mathit{claves}(\mathit{casas}(s)) = \emptyset \ \wedge \ \mathit{claves}(cs) \cap \mathit{claves}(\mathit{comercios}(s)) = \emptyset \end{array} \right\}$

unir : SimCity a $\times$ SimCity b $\longrightarrow$ SimCity
$\left\{ \begin{array}{l} \text{a} \neq b \wedge_{\text{L}} \mathit{OcupPorRios?}((\mathit{claves}(\mathit{casas}(b)) \cup \text{claves(comercios(b))}, \text{mapa(a)}) \wedge_{\text{L}} \\ \text{OcupPorRios?((claves(casas(a)) } \cup \text{ claves(comercios(a)), mapa(b))} \wedge_{\text{L}} \\ \text{posNivelMax(a)} \cap \text{posNivelMax(b)} = \emptyset \end{array} \right\}$

axiomas $\forall s, s'$: simcity, $\forall cs$: dicc(Pos, Construccion)

mapa(iniciar(m)) $\equiv$ m
mapa(avanzarTurno(s,cs)) $\equiv$ $\mathit{mapa}(s)$
mapa(unir(a, b)) $\equiv$ $\mathit{crear}((\mathit{horizontales}(\mathit{mapa}(a)) \cup \mathit{horizontales}(\mathit{mapa}(b))), (\mathit{verticales}(\mathit{mapa}(a)) \cup \mathit{verticales}(\mathit{mapa}(b))))$
casas(iniciar(m)) $\equiv$ $\emptyset$

casas(avanzarTurno(s,cs)) $\equiv$ $ActCasas(filtrar(casa, cs), sumarNivel(claves(casas(s)), casas(s)))$
casas(unir(a, b)) $\equiv$ $mergeCasas(claves(casas(b)), casas(a), casas(b))$
comercios(iniciar(m)) $\equiv$ $\emptyset$
comercios(avanzarTurno(s,cs)) $\equiv$ $ActComercios(filtrar(comercio, cs), casas(s),$
$sumarNivel(claves(comercios(s)), comercios(s)))$
comercios(unir(a, b)) $\equiv$ $unionComercial(casas(a), casas(b), comercios(a), comercios(b))$
popularidad(iniciar(m) $\equiv$ 0
popularidad(avanzarTurno(s, cs)) $\equiv$ $popularidad(s)$
popularidad(unir(a, b)) $\equiv$ $popularidad(a) + popularidad(b) + 1$
turnos(iniciar(m)) $\equiv$ 0
turnos(avanzarTurno(s, cs)) $\equiv$ $turnos(s) + 1$
turnos(unir(a, b)) $\equiv$ **if** $turnos(a) > turnos(b)$ **then** $turnos(a)$ **else** $turnos(b)$ **fi**

otras operaciones

OcupPorRios? : conj(Pos) $\times$ Mapa $\longrightarrow$ Bool
OcupPorRios?(c, m) $\equiv$ **if** $vacio?(c)$**then** $true$ **else** $(\pi_1(dameUno(c)) \notin verticales(m) \wedge_L$
$\pi_2(dameUno(c)) \notin horizontales(m)) \wedge_L OcupPorRios?(sinUno(c), m)$ **fi**
unirDicc : dicc(Pos $\times$ Nivel)a $\times$ dicc(Pos $\times$ Nivel)b $\longrightarrow$ dicc(Pos, Nivel)
$\{claves(a) \cap claves(b) = \emptyset\}$
unirDicc(a, b) $\equiv$ **if** $vacio?(claves(a))$**then** b **else**
$definir(dameUno(claves(a)), obtener(dameUno(claves(a), d)),$
$unirDicc(d, borrar(dameUno(claves(a)), a)))$**fi**
posNivelMax : SimCity $\longrightarrow$ conj(Pos)
posNivelMax(s) $\equiv$ $posConNivelN(unirDicc(casas(s), comercios(s)),$
$mayorNivel(unirDicc(casas(s), comercios(s))))$
posConNivelN : dicc(Pos $\times$ Nivel) $\times$ Nivel $\longrightarrow$ conj(Pos)
posConNivelN(d,n) $\equiv$ **if** $vacio?(claves(d))$**then** $\emptyset$ **else**
if$obtener(dameUno(claves(d)), d) = n$ **then**
$ag(dameUno(claves(d)), posConNivelN(borrar(dameUno(claves(d)), d), n))$
else $posConNivelN(borrar(dameUno(claves(d)), d), n)$**fi fi**
mayorNivel : dicc(Pos $\times$ Nivel) $\longrightarrow$ Nivel
mayorNivel(d) $\equiv$ **if** $vacio?(claves(d))$**then** 0 **else**
$max(obtener(dameUno(claves(d)), d),$
$mayorNivel(borrar(dameUno(claves(d)), d))$ **fi**
ActCasas : conj(pos) $\times$ dicc(pos $\times$ nivel) $\longrightarrow$ dicc(pos,nivel)
ActCasas(c, d) $\equiv$ **if** $vacio?(c)$ **then** d **else**
$ActCasas(sinUno(c), definir(dameUno(c), 0, d))$ **fi**
sumarNivel : conj(pos) $\times$ dicc(pos $\times$ nivel) $\longrightarrow$ dicc(pos, nivel)
sumarNivel(c, d) $\equiv$ **if** $vacio?(c)$ **then** d **else**
$sacarTurno(sinUno(c), definir(dameUno(c), obtener(dameUno(c), d)+1, d))$ **fi**
filtrar : construccion $\times$ dicc(pos $\times$ construccion) $\longrightarrow$ conj(pos)
filtrar(str, d) $\equiv$ **if** $vacio?(d)$ **then** $\emptyset$ **else**
if$obtener(dameUno(claves(d)), d) = str$ **then**
$ag(dameUno(claves(d)), filtrar(str, borrar(dameUno(claves(d)), d))$ **else**
$filtrar(str, borrar(dameUno(claves(d)), d))$**fi fi**
mergeCasas : conj(pos) $\times$ dicc(pos $\times$ nivel) $\times$ dicc(pos $\times$ nivel) $\longrightarrow$ dicc(pos, nivel)
mergeCasas(c, A, B) $\equiv$ **if** $vacio?(c)$ **then** A **else**
if$def?(dameUno(c), A)$ **then**
$mergeCasas(sinUno(c), A, B)$ **else**
$mergeCasas(sinUno(c), definir(dameUno(c), obtener(dameUno(c), B), A), B)$ **fi fi**
ActComercios : conj(pos) $\times$ dicc(pos $\times$ nivel) $\times$ dicc(poss $\times$ nivel) $\longrightarrow$ dicc(pos, nivel)
ActComercios(c, ca, co) $\equiv$ **if** $vacio?(c)$ **then** co **else if**$hayCasaCerca(dameUno(c), claves(ca))$ **then**
$ActComercios(sinUno(c), ca, definir(dameUno(c),$
$maxNivelCerca(dameUno(c), claves(ca), ca), co)$ **else**
$ActComercios(dameUno(c), ca, definir(dameUno(c), 0, co))$ **fi fi**
hayCasaCerca : pos $\times$ conj(pos) $\longrightarrow$ bool

hayCasaCerca(p, c) $\equiv$ **if** $vacio?(c)$ **then** $False$ **else if** $distManhattan(p, dameUno(c)) \leq 3$ **then** True **else** hayCasaCerca(p, sinUno(c)) **fi fi**
distManhattan : pos × pos $\longrightarrow$ nat
distManhattan(p_1, p_2) $\equiv$ $|\pi_1(p_1) - \pi_1(p_2)| + |\pi_2(p_1) - \pi_2(p_2)|$
maxNivelCerca : pos × conj(pos) × dicc(pos × nivel) $\longrightarrow$ nivel
maxNivelCerca(p, c, d) $\equiv$ **if** $vacio?(c)$ **then** 0 **else if** $distManhattan(p, dameUno(c)) \leq 3$ **then** max(obtener(dameUno(c), d), maxNivelCerca(p, sinUno(c), d)) **else** maxNivelCerca(p, sinUno(c), d) **fi fi**
unionComercial : dicc(pos × nivel) × dicc(pos × nivel) × dicc(pos × nivel) × dicc(pos × nivel) $\longrightarrow$ dicc(pos, nivel)
unionComercial(ca_a, ca_b, co_a, co_b) $\equiv$ $ultimoEslabon(claves(co_b), claves(ca_a) \cup$ claves(co_a), $predominaCasa(claves(co_a), ca_b, co_a), co_b)$
predominaCasa : conj(pos) × dicc(pos × nivel) × dicc(pos × nivel) $\longrightarrow$ dicc(pos, nivel)
predominaCasa(c, ca_b, co_a) $\equiv$ **if** $vacio?(c)$ **then** co_a **else if** $dameUno(c) \in claves(ca_b)$ **then** $predominaCasa(sinUno(c), ca_b, borrar(dameUno(c), co_a))$ **else** $predominaCasa(sinUno(c), ca_b, co_a)$ **fi fi**
ultimoEslabon : conj(pos) × conj(pos) × dicc(pos × nivel) × dicc(pos × nivel) $\longrightarrow$ dicc(pos, nivel)
ultimoEslabon(c, c_a, co_a, co_b) $\equiv$ **if** $vacio?(c)$ **then** co_a **else if** $dameUno(c) \in c_a$ **then** $ultimoEslabon(sinUno(c), c_a, co_a, co_b)$ **else** $ultimoEslabon(sinUno(c), c_a,$ $definir(dameUno(c), obtener(dameUno(c), co_b), co_a), co_b)$ **fi fi**
turnos : SimCity $\longrightarrow$ Nat

Comentario : Las decisiones (casa vs. casa, comercio vs. comercio o casa vs. comercio) se toman de la siguiente manera. Si hacemos la union, unir(a,b) entonces casas(a) > casas(b) > comercios(a) > comercios(b).

Fin TAD

NOMBRE es String

TAD SERVIDOR

igualdad observacional

$$(\forall s_1, s_2 \text{: Servidor})\ (s_1 =_{obs} s_2 \iff \begin{pmatrix} \text{partidas}(s_1) =_{obs} \text{partidas}(s_2) \wedge_L \\ (\forall n \text{: nombre})(n \in partidas(s_1) \Rightarrow_L \\ \text{partidaAsociada}(n, s_1) =_{obs} \text{partidaAsociada}(n, s_2)) \wedge_L \\ \text{modificables(s1)} =_{obs} \text{modificables(s2)} \end{pmatrix})$$

géneros Servidor

exporta Servidor, generadores, observadores

usa conj(Nat), Nat, SimCity, Mapa, nombres as int

observadores básicos

partidas : Servidor $\longrightarrow$ conj(nombres)
modificables : Servidor $\longrightarrow$ conj(nombres)
partidaAsociada : nombre n × servidor s $\longrightarrow$ simCity {n $\in$ $partidas(s)$}

generadores

iniciar : $\longrightarrow$ Servidor
agregar : nombre n × simcity sc × servidor s $\longrightarrow$ Servidor {n $\notin$ $partidas(s)$}
unir : nombre a × nombre b × Servidor s $\longrightarrow$ Servidor

$$\left\{\begin{array}{l}(\text{a} \in modificable(s) \land a \in partidas(s) \land b \in partidas(s) \land a \neq b) \\ \land_{\text{L}} \\ OcupPorRios?((claves(casas(partidaAsociada(b,s))) \cup \\ claves(comercios(partidaAsociada(b,s)))), mapa(partidaAsociada(a,s))) \\ \land_{\text{L}} \\ OcupPorRios?((claves(casas(partidaAsociada(a,s))) \cup \\ claves(comercios(partidaAsociada(a,s)))), mapa(partidaAsociada(b,s))) \\ \land_{\text{L}} \\ posNivelMax(partidaAsociada(a,s)) \cap \\ posNivelMax(partidaAsociada(b,s)) = \emptyset \end{array}\right\}$$

avanzarTurno : nombre n × dicc(Pos × Construccion) cs × Servidor $\longrightarrow$ Servidor

$$\left\{\begin{array}{l}\text{n} \in partidas(s) \land_{\text{L}} \#claves(cs) \geq 1 \land_{\text{L}} \\ OcupPorRios?(claves(cs), mapa(partidaAsociada(n))) \\ \land_{\text{L}} claves(cs) \cap claves(casas(partidaAsociada(n))) = \emptyset \land_{\text{L}} \\ claves(cs) \cap claves(comercios(partidaAsociada(n))) = \emptyset \end{array}\right\}$$

otras operaciones

axiomas $\forall s$: servidor,$\forall sc$: simcity,$\forall n, n_1, n_2$: nombre, $\forall d$: dicc(Pos, Construccion)

partidas(iniciar(s))	$\equiv$	$\emptyset$
partidas(agregar(n,sc,s))	$\equiv$	Ag(n, partidas(s))
partidas(unir(n_1, n_2, s)	$\equiv$	partidas(s)
partidas(avanzarTurno(n, d, s)	$\equiv$	partidas(s)
modificables(iniciar(s))	$\equiv$	$\emptyset$
modificables(agregar(n,sc,s))	$\equiv$	Ag(n,modificables(s))
modificables(unir(n_1, n_2, s)	$\equiv$	modificables(s) - n_2
modificables(avanzarTurno(n, d, s)	$\equiv$	modificables(s)
partidaAsociada(n, agregar(n_1, sc, s)	$\equiv$	**if** n=n_1 **then** sc **else** $partidaAsociada(n,s)$ **fi**
partidaAsociada(n, unir(n_1, n_2, s)	$\equiv$	partidaAsociada(n, s)
partidaAsociada(n, avanzarTurno(n_1, d, s)	$\equiv$	partidaAsociada(n, s)

Fin TAD

2. Módulos de referencia

2.1. Módulo Mapa

Interfaz

se explica con: MAPA

géneros: `mapa`

Operaciones básicas de mapa

CREAR(**in** hs: `conjLineal(Nat)`,**in** vs: `conjLineal(Nat)`) $\rightarrow$ res : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} mapa(hs, vs)\}$
Complejidad: $O(copy(hs), copy(vs))$
Descripción: crea un mapa

HORIZONTALES(**in** m: `mapa`) $\rightarrow$ res : `conjLineal(Nat)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} horizontales(m)\}$
Complejidad: $O(1)$
Descripción: devuelve el conjunto de rios horizontales

VERTICALES(**in** m: `mapa`) $\rightarrow$ res : `conjLineal(Nat)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} verticales(m)\}$
Complejidad: $O(1)$)
Descripción: devuelve el conjunto de rios verticales

Representación

Representación de mapa

Un mapa contiene rios infinitos horizontales y verticales. Los rios se representan como conjuntos lineales de naturales que indican la posición en los ejes de los ríos.

`mapa` **se representa con** `estr`

donde `estr` es `tupla`(*horizontales*: `conj(Nat)`, *verticales*: `conj(Nat)`)

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\iff$ true

Abs : estr m $\longrightarrow$ mapa {Rep(m)}
Abs(m) $\equiv$ horizontales(m) = $estr$.horizontales $\wedge$ verticales(m) = $estr$.verticales

Algoritmos

---

crear(**in** hs: `conj(Nat)`, **in** vs: `conj(Nat)`) $\rightarrow$ res : estr
1: $estr.horizontales \leftarrow hs$
2: $estr.verticales \leftarrow vs$ **return** estr

Complejidad: $O(copy(hs) + copy(vs))$

---

horizontales(**in** m: `mapa`) $\rightarrow$ res : conj(nat)
return $estr.horizontales$ Complejidad: $O(1)$

---

verticales(**in** m : `mapa`) → res : conj(nat)
return *estr.verticales* Complejidad: $O(1)$

2.2. Módulo Simcity

Interfaz

se explica con: SIMCITY

géneros: `Simcity`

Operaciones básicas de Simcity

INICIAR(**in** m : `Mapa`) → res : `Simcity`
Pre ≡ {true}
Post ≡ $\{res =_{\text{obs}} inciar(m)\}$
Complejidad: $O(1)$
Descripción: inicia un SimCity

AVANZARTURNO(**in/out** s : `SimCity`,**in** cs : `dicc(Pos, construccion)`)
Pre ≡ $\{s =_{\text{obs}} s_0 \land \#claves(cs) \geq 1 \land\ OcupPorRios?(claves(cs), mapa(s)) \land$
$claves(cs) \cap claves(casas(s)) = \emptyset\ \land\ claves(cs) \cap claves(comercios(s)) = \emptyset\}$
Post ≡ $\{s =_{\text{obs}} avanzarTurno(s_0, cs)\}$
Complejidad: $O(1)$
Descripción: pasa un turno en un Simcity

UNIR(**in/out** a : `SimCity`, **in** b : `SimCity`)
Pre ≡ $\{a =_{\text{obs}} a_0 \land a \neq b \land OcupPorRios?((claves(casas(b))\cup$ claves(comercios(b)), mapa(a)) $\land$
$OcupPorRios?((claves(casas(a))\cup$ claves(comercios(a)), mapa(b)) $\land$
$posicionesNivelMaximo(a)\cap$ posicionesNivelMaximo(b) $= \emptyset\}$
Post ≡ $\{a =_{\text{obs}} unir(a_0, b)\}$
Complejidad: $O(1)$
Descripción: une dos SimCity

MAPA(**in** sc: `SimCity`) → res : `Mapa`
Pre ≡ {true}
Post ≡ $\{res =_{\text{obs}} mapa(sc)\}$
Complejidad: $O(1)$
Descripción: devuelve el mapa utilizado del SimCity que se le pasa

CASAS(**in** sc: `SimCity`) → res : `dicc(pos, nivel)`
Pre ≡ {true}
Post ≡ $\{res =_{\text{obs}} casas(sc)\}$
Complejidad: $O(1)$
Descripción: devuelve un diccionario de las casas de un SimCity, asociando una posicion a un nivel

COMERCIOS(**in** sc: `SimCity`) → res : `dicc(pos, nivel)`
Pre ≡ {true}
Post ≡ $\{res =_{\text{obs}} comercios(sc)\}$
Complejidad: $O(1)$
Descripción: devuelve un diccionario de los comercios de un SimCity, asociando una posicion a un nivel

POPULARIDAD(**in** sc: `SimCity`) → res : `Nat`
Pre ≡ {true}
Post ≡ $\{res =_{\text{obs}} popularidad(sc)\}$
Complejidad: $O(1)$
Descripción: devuelve la popularidad de un SimCity

TURNOS(**in** sc: `SimCity`) → res : `Nat`
Pre ≡ {true}
Post ≡ $\{res =_{\text{obs}} turnos(sc)\}$

Complejidad: $O(1)$
Descripción: devuelve el turno de un SimCity

Representación

Representación de SimCity

Un SimCity hereda los rios utilizados por el mapa usado para generarlo. A su vez contiene un diccionario de casas y uno de comercios, cada uno indicando la posicion de cada edificacion con su respectivo nivel. Por ultimo, cada SimCity cuenta con una popularidad, que representa la cantidad de uniones a las que este mapa fue sujeto. Se decidió que las decisiones (casa vs. casa, comercio vs. comercio o casa vs. comercio) se toman de la siguiente manera. Si hacemos la union, unir(a,b) entonces casas(a) > casas(b) > comercios(a) > comercios(b).

`SimCity` **se representa con** `estr`

donde `estr` es `tupla`(*mapa*: `Mapa`, *turno*: `Nat`, *popularidad*: `Nat`, *casas*: `diccLineal(Pos,Nivel)`, *comercios*: `diccLineal(Pos,Nivel)`, *unidos*: `lista(puntero(SimCity))`)

Rep : estr e $\longrightarrow$ bool

$$\begin{aligned}\text{Rep}(e) \equiv\ & \text{true} \iff (\forall p : Pos)(p \in (claves(e.casas) \cup claves(e.comercios)) \Rightarrow_{\text{L}} \pi_1(p) \notin verticales(e.mapa) \\ & \wedge\ \pi_2(p) \notin horizontales(e.mapa)\ \wedge (e.turno \geq obtener(p, unirDicc(e.casas, e.comercios)))) \wedge \\ & (claves(e.casas) \cap claves(e.comercios) = \emptyset)\ \wedge \\ & ((e.turno \neq 0 \wedge\ (\#claves(e.casas) + \#claves(e.comercios)) > 0) \vee \\ & (e.turno = 0 \wedge (\#claves(e.casas) + \#claves(e.comercios) = 0))) \wedge \\ & e.popularidad = \textstyle\sum_{k=0}^{long(e.unidos)} *(e.unidos[k]).popularidad + long(e.unidos) \wedge \\ & ((\forall p, p' : puntero(SimCity))(p \in e.unidos) \Rightarrow_{\text{L}} \\ & OcupPorRios?((claves(e.casas) \cup claves(e.comercios), *(p).mapa) \wedge \\ & OcupPorRios?((claves(*(p).casas) \cup claves(*(p).comercios), e.mapa) \wedge \\ & posNivelMax(e) \cap posNivelMax(*(p)) = \emptyset\ \wedge e.turno \geq *(p).turno \wedge \\ & OcupPorRios?((claves(*(p').casas) \cup claves(*(p').comercios), *(p).mapa) \wedge \\ & OcupPorRios?((claves(*(p).casas) \cup claves(*(p).comercios), *(p').mapa) \wedge \\ & posNivelMax(*(p')) \cap posNivelMax(*(p)) = \emptyset) \wedge Rep(*(p))\end{aligned}$$

Comentario: Llamamos recursivamente al Rep ya que para todo puntero y a su vez para todo subpuntero de subpuntero y asi siguiente, se debe cumplir todo lo anterior.

Abs : estr e $\longrightarrow$ simCity {Rep(e)}

$$\begin{aligned}\text{Abs}(e) \equiv\ & \textbf{if } long(e.unidos) = 0 \textbf{ then} \\ & \quad \textbf{if} \neg vacio?(claves(e.casas))\ \wedge\ \neg vacio?(claves(e.comercios)) \textbf{ then} \\ & \quad\quad avanzarTurno(Abs(<e.mapa, e.turno - 1, e.popularidad, \\ & \quad\quad sinNivelCero(e.casas, diccNivelCero(e.casas, e.comercios), casa), \\ & \quad\quad sinNivelCero(e.comercios, diccNivelCero(e.casas, e.comercios), comercio), e.unidos>), \\ & \quad\quad diccNivelCero(e.casas, e.comercios)) \\ & \quad \textbf{else} \\ & \quad\quad iniciar(e.mapa) \\ & \quad \textbf{fi} \\ & \textbf{else} \\ & \quad unir(Abs(<e.mapa, e.turno, e.popularidad - 1, e.casas, e.comercios, com(e.unidos)>), *(ult(e.unidos))) \\ & \textbf{fi}\end{aligned}$$

diccLineal(pos, construccion)

diccNivelCero : diccLineal(pos $\times$ nivel) $\times$ diccLineal(pos $\times$ nivel) $\longrightarrow$ diccLineal(pos, construccion)

diccNivelCero(ca, co) $\equiv$ $crearDiccCero(claves(ca), ca, claves(co), co)$

crearDiccCero : conjLineal(pos) $\times$ diccLineal(pos $\times$ nivel) $\times$ conjLineal(pos) $\times$ diccLineal(pos $\times$ nivel) $\longrightarrow$ diccLineal(pos, construccion)

crearDiccCero(c_a, ca, c_o, co) $\equiv$ **if** $\neg vacio?(c_a)$ **then**
if $obtener(dameUno(c_a), ca) = 0$ **then**
$definir(dameUno(c_a), casa, crearDiccCero(sinUno(c_a), ca, c_o, co))$
else
$crearDiccCero(sinUno(c_a), ca, c_o, co)$
fi
else
if $\neg vacio?(c_o)$ **then**
if $obtener(dameUno(c_o), co) = 0$ **then**
$definir(dameUno(c_o), comercio, crearDiccCero(c_a, ca, sinUno(c_o), co))$
else
$crearDiccCero(c_a, ca, sinUno(c_o), co)$
fi
else
$vacio()$
fi

sinNivelCero : diccLineal(pos × nivel) × diccLineal(pos × nivel) × construccion $\longrightarrow$ diccLineal(pos, nivel)

sinNivelCero(d, cs, str) $\equiv$ **if** $vacio?(claves(cs))$ **then**
d
else
if $obtener(dameUno(claves(cs)), cs) = str$ **then**
$sinNivelCero(borrar(dameUno(claves(cs)), d),$
$borrar(dameUno(claves(cs)), cs), str)$
else
$sinNivelCero(d, borrar(dameUno(claves(cs)), cs), str)$
fi
fi

Iniciar(**in** m : `Mapa`) $\rightarrow$ res : estr

1: $estr.mapa \leftarrow m$
2: $estr.turno \leftarrow 0$
3: $estr.popularidad \leftarrow 0$
4: $estr.casas \leftarrow vacio()$
5: $estr.comercios \leftarrow vacio()$
6: $estr.unidos \leftarrow vacia()$ **return** estr

Complejidad: $O(copy(m))$

---

avanzarTurno(**in**/**out** s : `SimCity`,**in** cs : `dicc(Pos, construccion)`)

```
1: casasFinal = casas(s)
2: comerciosFinal = comercios(s)
3: for i in claves(casasFinal) :
4:   definirRapido(casasFinal, i, obtener(casasFinal, i) + 1)
5: for j in claves(comerciosFinal) :
6:   definirRapido(comerciosFinal, i, obtener(comerciosFinal, i) + 1)
7:
8: i ← CrearIt(claves(cs))
9: while haySiguiente(i)
10:  if obtener(i, cs) == casa then
11:    definirRapido(i, 0, casasFinal)
12:  else definirRapido(i, 0, comerciosFinal)
13:  avanzar(i)
14:
15: s.casas = casasFinal
16: s.comercios = comerciosFinal
17: s.turno = s.turno + 1
```

Complejidad: $O(copy(claves(cs)) + copy(casas(cs)) + copy(comercios(cs)))$

---

unir(**in**/**out** a : `SimCity`, **in** b : `SimCity`)

```
1: a.popularidad ← a.popularidad + b.popularidad + 1
2: a.turno ← max(a.turno, b.turno)
3: a.unidos ← agregarAtras(a.unidos, b)
```

Complejidad: $O(1)$

---

mapa(**in** sc : `SimCity`) → res : Mapa

```
1: res = sc.mapa
2: for i = 0 to long(sc.unidos) :
3:   unirMapa(sc.unidos[i], res), donde n es la cantidad de SimCities. return res
```

Complejidad: $\sum_{i=0}^{n} copy(n.mapa)$

---

unirMapa(**in** sc : `SimCity`,**in**/**out** $mapaFinal$: `mapa`)

```
1: if(long(sc.unidos) = 0) :
2:   mapaFinal.horizontales ← mapaFinal.horizontales ∪ sc.mapa.horizontales
3:   mapaFinal.verticales ← mapaFinal.verticales ∪ sc.mapa.verticales
4: else :
5:   for j = 0 to long(cs.unidos) :
6:     unirMapa(cs.unidos[j], mapaFinal)
```

Complejidad: $O(\sum_{i=0}^{n} copy(n.mapa))$, donde n es la cantidad de SimCities.

---

casas(**in** sc : `SimCity`) → res : dicc(pos, nivel)

```
1: res = copy(cs.casas)
2: for i = 0 to long(cs.unidos) :
3:   agregarCasasUnion(sc.unidos[i], res) return res
```

Complejidad: $O(\sum_{i=0}^{n} copy(n.casas))$, donde n es la cantidad de SimCities.

---

agregarCasasUnion(**in** *sc*: `SimCity`, **in/out** *casasFinal*: `dicc(Pos, Nivel)`)

```
1: if(long(sc.unidos) = 0) :
2:   for i in claves(sc.casas) :
3:     if not(definido?(casasFinal, i)) :
4:       definirRapido(i, obtener(i, cs.casas), casasFinal)
5: else :
6:   for j = 0 to long(cs.unidos) :
7:     agregarCasasUnion(cs.unidos[j], casasFinal)
```

Complejidad: $O(\sum_{i=0}^{n} copy(n.casas))$, donde n es la cantidad de SimCities.

comercios(**in** *sc*: `SimCity`) → *res* : dicc(pos, nivel)

```
1: res = copy(cs.comercios)
2: casas = casas(cs)
3: for i = 0 to long(cs.unidos) :
4:   agregarComerciosUnion(sc.unidos[i], casas, res)
5:
6: for i in claves(res) :
7:   maxNivelCercano = 0
8:   for j in claves(casas) :
9:     if(casas[j] > res[i] && |i − j| ≤ 3 && casas[j] > maxNivelCercano) :
10:      maxNivelCercano = casas[j]
11:  if(maxNivelCercano ≠ 0) :
12:    res[i] = maxNivelCercano
13: return res
```

Complejidad: $O(\sum_{i=0}^{n} copy(n.comercios) + copy(n.casas))$, donde n es la cantidad de SimCities.

agregarComerciosUnion(**in** *sc*: `SimCity`,**in** *casas*: `dicc(Pos, Nivel)`, **in/out** *comerciosFinal*: `dicc(Pos, Nivel)`)

```
1: if(long(sc.unidos) = 0) :
2:   for i in claves(sc.comercios) :
3:     if (not definido?(casas, i) and not definido?(comerciosFinal, i)) :
4:       definirRapido(i, obtener(i, cs.comercios), comerciosFinal)
5: else :
6:   for j = 0 to long(cs.unidos) :
7:     agregarComerciosUnion(cs.unidos[j], casas, comerciosFinal)
```

Complejidad: $O(\sum_{i=0}^{n} copy(n.comercios))$, donde n es la cantidad de SimCities.

popularidad(**in** *sc*: `SimCity`) → *res* : Nat

```
1: res ← sc.popularidad return res
```

Complejidad: $O(1)$

turnos(**in** *sc*: `SimCity`) → *res* : Nat

```
1: res ← sc.turno return res
```

Complejidad: $O(1)$

2.3. Módulo Servidor

Interfaz

se explica con: SERVIDOR

géneros: `Servidor`

Operaciones básicas de Servidor

INICIAR() $\rightarrow$ *res* : `Servidor`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} vacio$}
Complejidad: $O(1)$
Descripción: crea un servidor vacio

AGREGAR(**in** n : `nombre`,**in** sc: `SimSity`, **in/out** s: `Servidor`)
Pre $\equiv$ {$s =_{\text{obs}} s_0 \land n \notin partidas(s)$}
Post $\equiv$ {$s =_{\text{obs}} Agregar(n, sc, s_0)$}
Complejidad: $O(|n|)$, donde n es el nombre mas largo de los SimCitys
Descripción: agrega un SimCity al servidor

UNIR(**in** a : `nombre`,**in** b : `nombre`, **in/out** s : `Servidor`)
Pre $\equiv$ {$s =_{\text{obs}} s_0 \land (a \in modificable(s) \land a \in partidas(s) \land b \in partidas(s) \land a \neq b)$
$\land$
$OcupPorRios?((claves(casas(partidaAsociada(b, s))) \cup$
$claves(comercios(partidaAsociada(b, s)))), mapa(partidaAsociada(a, s)))$
$\land$
$OcupPorRios?((claves(casas(partidaAsociada(a, s))) \cup$
$claves(comercios(partidaAsociada(a, s)))), mapa(partidaAsociada(b, s)))$
$\land$
$posicionesNivelMaximo(partidaAsociada(a, s)) \cap$
$posicionesNivelMaximo(partidaAsociada(b, s)) =_{\text{obs}} \emptyset$}
Post $\equiv$ {$s =_{\text{obs}} unir(a, b, s_0) \land b \notin modificables(s)$}
Complejidad: $O(|n|)$, donde n es el nombre mas largo de los SimCitys
Descripción: une dos SimCitys con el criterio, casas(a) > casas(b) > comercios(a) > comercios(b), mas los criterios preestablecidos por las catedra.
Aliasing: Cuando se redefine un elemento del diccionario, su significado se pasa por referencia

AVANZARTURNO(**in** n : `nombre`,**in** cs : `dicc(pos, construccion)` , **in/out** *Servidor* :)
Pre $\equiv$ {$s =_{\text{obs}} s_0 \land n \in partidas(s) \land \#claves(cs) \geq 1 \land$
$OcupPorRios?(claves(cs), mapa(partidaAsociada(n)))$
$\land claves(cs) \cap claves(casas(partidaAsociada(n))) =_{\text{obs}} \emptyset \land$
$claves(cs) \cap claves(comercios(partidaAsociada(n))) =_{\text{obs}} \emptyset$}
Post $\equiv$ {$s =_{\text{obs}} avanzarTurno(n, cs, s_0)$}
Complejidad: $O(|n|)$, donde n es el nombre mas largo de los SimCitys
Descripción: avanza un turno en el SimCity que se le pasa y se agregan las respectivas construcciones
Aliasing: Cuando se redefine un elemento del diccionario, su significado se pasa por referencia

PARTIDAS(**in** s: `Servidor`) $\rightarrow$ *res* : `conj(nombres)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} partidas(s)$}
Complejidad: $O(1)$
Descripción: devuelve el conjunto de partidas

PARTIDAASOCIADA(**in** n: `nombre`,**in** s: `Servidor`) $\rightarrow$ *res* : `Simcity`
Pre $\equiv$ {n $\in partidas(s)$}
Post$\equiv$ {$res =_{\text{obs}} partidaAsociada(n, s)$}
Complejidad: $O(|n|)$, donde n es el nombre mas largo de los SimCitys
Descripción: devuelve la partida asociada a determinado nombre y servidor

Representación

Representación de Servidor

Un Servidor almacena las partidas de los distintos jugadores, cada una con su respectivo nombre; como asi tambien aquellas partidas que siguen siendo modificables. Tambien puede realizar ciertas operaciones sobre

ellos.

`servidor` **se representa con** `diccTrie(Clave,significado)`

donde `significado` es `tupla(`*simcity*: `SimCity`, *modif*: `Bool`)

Rep : diccTrie(clave × significado) $\longrightarrow$ bool
Rep(e) $\equiv$ true

Abs : diccTrie(clave × significado) d $\longrightarrow$ Servidor {Rep(d)}
Abs(d) $\equiv$ **if** $vacio?(d.claves)$ **then** $iniciar()$ **else**
$agregar(dameUno(d.claves), obtener(dameUno(d.claves), d), Abs(borrar(dameUno(d.claves), d))$ **fi**

Iniciar() $\rightarrow$ res : Servidor
1: $res \leftarrow Nuevo()$ **return** res

Complejidad: $O(1)$

---

agregar(**in** n : `nombre`, **in** sc : `SimCity`, **in/out** s : `servidor`)
1: $definir(n, < sc, True >, s)$

Complejidad: $O(|n|)$

---

unir(**in** a : `nombre`,**in** b : `nombre`, **in/out** s : `Servidor`)
1: $definir(a, < unir(partidaAsociada(a, s), partidaAsociada(b, s)), True >, s)$
2: $definir(b, < partidaAsociada(b, s)), False >, s)$
3:

Complejidad: $O(max(|a|, |b|))$

---

avanzarTurno(**in** n : `nombre`,**in** cs : `dicc(pos,construccion)`, **in/out** s : `servidor`)
1: $definir(n, < avanzarTurno(partidaAsociada(n, s)), True >, s)$

Complejidad: $O(|n * |Claves(cs)||)$

---

partidas(**in** s : `Servidor`) $\rightarrow$ res : conj(Nombres)
1: $res \leftarrow Claves(s)$ **return** res

Complejidad: $O(1)$

---

partidaAsociada(**in** n : `Nombre`, **in** s : `Servidor`) $\rightarrow$ res : SimCity
1: $res \leftarrow \pi_1(Obtener(n, s))$ **return** res

Complejidad: $O(|n|)$

Operaciones básicas de diccTrie(Nombre, Simcity)

NUEVO() $\rightarrow$ res : `diccTrie(Nombre, Simcity)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} vacio()$}
Complejidad: $O(1)$
Descripción: crea un diccionario vacio

DEFINIR(**in** n: `Nombre`, **in** sc: `Simcity`, **in** d: `diccTrie(Nombre, Simcity)`) $\rightarrow$ res : `diccTrie(Nombre, Simcity)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} definir(n, sc, d)\}$
Complejidad: $O(|n|)$
Descripción: define un nuevo nombre con un SimCity asociado

CLAVES(**in** d: `diccTrie(Nombre,Simcity)`) $\rightarrow$ res : `ConjLineal(Nombre)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} claves(d)\}$
Complejidad: $O(1)$
Descripción: devuelve el conjunto de claves del diccionario

BORRAR(**in** n: `Nombre`, **in/out** d: `diccTrie(Nombre,Simcity)`)
Pre $\equiv$ $\{d =_{\text{obs}} d_0\}$
Post $\equiv$ $\{d =_{\text{obs}} borrar(n, d_0)\}$
Complejidad: $O(|n|)$
Descripción: borra un elemento del diccionario

OBTENER(**in** n: `Nombre`, **in** d: `diccTrie(Nombre, SimCity)`) $\rightarrow$ res : `SimCity`
Pre $\equiv$ $\{n \in claves(d)\}$
Post $\equiv$ $\{res =_{\text{obs}} obtener(n, d)\}$
Complejidad: $O(|n|)$
Descripción: obtiene la partida asociada a un nombre