DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP 1: Diseño

SimCity

1 de Junio de 2022 — Algoritmos y Estructuras de Datos II

Grupo 01 - hasTADlaVista, turno mañana

Integrante	LU	Correo electrónico
Lakowsky, Manuel	511/21	mlakowsky@gmail.com
Vekselman, Natán	338/21	natanvek11@gmail.com
Arienti, Federico	316/21	fa.arienti@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
http://www.exactas.uba.ar

Resúmen

El siguiente trabajo busca desarrollar el diseño de algunas de las estructuras de datos centrales al juego SimCity de 1989. Se buscará modelar e implementar el TAD *SimCity*, y los TADs satélite *Mapa* y *Servidor*, en concordancia con las interpretaciones y restricciones consignadas. En cada caso, se detallarán las motivaciones detrás de las soluciones propuestas.

Un SimCity consistirá, en términos generales, de un *Mapa* y un conjunto de *Construcciones* de tipo *casa* o *comercio*. El mismo permitirá la *union* con otras instancias del tipo, y deberá permitir conocer el *turno* y la *popularidad* de la partida, entendido éste último atributo como la cantidad de uniones que componen a la instancia.

Índice

1. Especificación **2**
1.1. Aliases . . . 2
1.2. Mapa . . . 2
1.3. SimCity . . . 3
1.4. Servidor . . . 6

2. Diseño **8**
2.1. Aliases . . . 8
2.2. Módulo Mapa . . . 8
2.2.1. Interfaz . . . 8
2.2.2. Representación . . . 9
2.2.3. Implementación . . . 10
2.3. Módulo SimCity . . . 11
2.3.1. Interfaz . . . 11
2.3.2. Representación . . . 13
2.3.3. Implementación . . . 18
2.4. Módulo Servidor . . . 22
2.4.1. Interfaz . . . 22
2.4.2. Representación . . . 24
2.4.3. Implementación . . . 25

1. Especificación

1.1. Aliases

TAD POS es TUPLA<x: NAT × y: NAT>

TAD CONSTRUCCIÓN es STRING

TAD NOMBRE es STRING

TAD NIVEL es NAT

1.2. Mapa

TAD MAPA

géneros Mapa

exporta Mapa, observadores, generadores, $\bullet + \bullet$, esRio

usa Nat, conj(α), Pos, Bool

igualdad observacional

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

observadores básicos

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

generadores

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

otras operaciones

$\bullet + \bullet$: Mapa × Mapa $\longrightarrow$ Mapa
esRio : Pos × Mapa $\longrightarrow$ Bool

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

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

otros ax. $\forall m1,\ m2$: Mapa, $\forall p$: Pos

$m1 + m2$ $\equiv$ crear(horizontales($m1$) $\cup$ horizontales($m2$), verticales($m1$) $\cup$ verticales($m2$))
esRio(p, $m1$) $\equiv$ $p.x \in$ verticales($m1$) $\vee$ $p.y \in$ horizontales($m1$)

Fin TAD

1.3. SimCity

TAD SimCity

géneros SimCity

exporta SimCity, observadores, generadores, turnos, $\bullet \cup_{dicc} \bullet$

usa Mapa, Nat, Pos, Construcción, Nivel, dicc($\alpha \times \beta$)

igualdad observacional

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

observadores básicos

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

generadores

iniciar	: Mapa	$\longrightarrow$	SimCity	
avanzarTurno	: SimCity s $\times$ dicc(Pos $\times$ Construccion) cs	$\longrightarrow$	SimCity	{avanzarValido(s, cs)[1]}
unir	: SimCity a $\times$ SimCity b	$\longrightarrow$	SimCity	{unirValido(a, b)[1]}

otras operaciones

turnos	: SimCity	$\longrightarrow$	Nat
construcciones	: SimCity	$\longrightarrow$	dicc(Pos $\times$ Nivel)
agCasas	: dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Construcción)	$\longrightarrow$	dicc(Pos $\times$ Nivel)
agComercios	: dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Construcción)	$\longrightarrow$	dicc(Pos $\times$ Nivel)
nivelComercio	: Pos $\times$ Nat $\times$ dicc(Pos $\times$ Nivel)	$\longrightarrow$	Nat
distManhattan	: Pos $\times$ Pos	$\longrightarrow$	Nat
$\bullet \cup_{dicc} \bullet$	: dicc($\alpha \times \beta$) $\times$ dicc($\alpha \times \beta$)	$\longrightarrow$	dicc($\alpha \times \beta$)
sacarRepes	: dicc(Pos $\times$ Construcción) $\times$ dicc(Pos $\times$ Construcción)	$\longrightarrow$	dicc(Pos $\times$ Construcción)
avanzarNivel	: dicc(Pos $\times$ Nivel)	$\longrightarrow$	dicc(Pos, Nivel)
unirConst	: dicc(Pos $\times$ Construccion) $\times$ dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Nivel)	$\longrightarrow$	dicc(Pos, Nivel)
manhatizar	: dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Nivel)	$\longrightarrow$	dicc(Pos, Nivel)

axiomas $\forall s,\ s'$: simcity, $\forall m$: Mapa, $\forall cs$: dicc(Pos $\times$ Construcción)

mapa(iniciar(m))	$\equiv$	m
mapa(avanzarTurno(s, cs))	$\equiv$	mapa(s)
mapa(unir(s, s'))	$\equiv$	mapa(s) + mapa(s')
casas(iniciar(m))	$\equiv$	vacio
casas(avanzarTurno(s, cs))	$\equiv$	agCasas(avanzarNivel(casas(s)), cs)
casas(unir(s, s'))	$\equiv$	unirConst(construcciones(s), casas(s), casas(s'))
comercios(iniciar(m))	$\equiv$	vacio
comercios(avanzarTurno(s, cs))	$\equiv$	manhatizar(casas(s), agComercios(avanzarNivel(comercios(s)), cs))
comercios(unir(s, s'))	$\equiv$	manhatizar(casas(unir(s, s')), unirConst(construcciones(s), comercios(s), comercios(s')))
popularidad(iniciar(m))	$\equiv$	0
popularidad(avanzarTurno(s, cs))	$\equiv$	popularidad(s)

popularidad(unir(s, s')) $\equiv$ popularidad(s) + 1 + popularidad(s')

turnos(iniciar(m)) $\equiv$ 0
turnos(avanzarTurno(s, cs)) $\equiv$ turnos(s) + 1
turnos(unir(s, s')) $\equiv$ **if** turnos(s) < turnos(s') **then** turnos(s') **else** turnos(s) **fi**

otros ax. $\forall s$: simcity, $\forall p$, q: Pos, $\forall cs$, cs': dicc(Pos × Construcción), $\forall cn$, cn': dicc(Pos × Nivel), $\forall d$, d': dicc($\alpha \times \beta$)

construcciones(s) $\equiv$ casas(s) $\cup_{dicc}$ comercios(s)

agCasas(cn, cs) $\equiv$ **if** vacio?(claves(cs)) **then**
cn
else if obtener(proximo, cs) = "*casa*" **then**
agCasas(definir(proximo, 1, cn), borrar(proximo, cs))
else
agCasas(cn, borrar(proximo, cs))
fi
donde proximo $\equiv$ dameUno(claves(cs))

agComercios(cn, cs) $\equiv$ **if** vacio?(claves(cs)) **then**
cn
else if obtener(proximo, cs) = "*comercio*" **then**
agComercios(definir(proximo, 1, cn), borrar(proximo, cs))
else
agComercios(cn, borrar(proximo, cs))
fi
donde proximo $\equiv$ dameUno(claves(cs))

nivelComercio(p, n, cn) $\equiv$ **if** vacio?(claves(cn)) **then**
n
else if distManhattan(p, proximo) $\leq 3 \wedge$ obtener(proximo, cn) $> n$ **then**
nivelComercio(p, obtener(proximo, cn), borrar(proximo, cn))
else
nivelComercio(p, n, borrar(proximo, cn))
fi
donde proximo $\equiv$ dameUno(claves(cn))

distManhattan(p, q) $\equiv$ **if** $p.x < q.x$ **then** $q.x$ - $p.x$ **else** $p.x$ - $q.x$ **fi**
+
if $p.y < q.y$ **then** $q.y$ - $p.y$ **else** $p.y$ - $q.y$ **fi**

$d \cup_{dicc} d'$ $\equiv$ **if** vacio?(claves(d')) **then**
d
else if $\neg$ def?(proximo, d) **then**
definir(proximo, obtener(proximo, d'), d) $\cup_{dicc}$ borrar(proximo, d')
else
$d \cup_{dicc}$ borrar(proximo, d')
fi
donde proximo $\equiv$ dameUno(claves(d'))

sacarRepes(cs, cs') $\equiv$ **if** vacio?(claves(cs)) **then**
cs'
else if def?(proximo, cs') **then**
sacarRepes(borrar(proximo, cs), borrar(proximo, cs'))
else
sacarRepes(borrar(proximo, cs), cs')
fi
donde proximo $\equiv$ dameUno(claves(cs))

1. definido en el apartado Definiciones Auxiliares de SimCity.
2. las funciones **agCasas** y **agComercios** agregan respectivamente al diccionario de casas/comercios las construcciones de entrada sin importar si esas posiciones ya se encontraban ocupadas o no. Esto no genera un problema en avanzar turno, por sus restricciones, pero si al unir SimCitys (ya que podrian haber colisiones). Para solucionar esto, sacarRepes quita del diccionario de entrada las posiciones ocupadas por construcciones ya establecidas. Es decir, como resolución al conflicto de colisiones, las construcciones que permanecen son las del SimCity original.

avanzarNivel(cs) ≡ **if** vacio?(claves(cs)) **then**
cs
else
definir(dameUno(claves(cs)),
obtener(dameUno(claves(cs)), cs) + 1,
avanzarNivel(borrar(dameUno(claves(cs))), cs))
fi

unirConst(cs, cn, cn') ≡ **if** vacio?(cn') **then**
cn
else if ¬ def?(proximo, cs) **then**
unirConst(cs,
definir(proximo, obtener(proximo, cn'), cn),
borrar(proximo, cn'))
else
unirConst(cs, cn, borrar(proximo, cn'))
fi
donde proximo ≡ dameUno(claves(cn'))

manhatizar(cn, cn') ≡ **if** vacio?(cn') **then**
vacio
else
definir(proximo,
nivelComercio(proximo, obtener(proximo, cn'), cn),
manhatizar(cn, borrar(proximo, cn')))
fi
donde proximo ≡ dameUno(claves(cn'))

Fin TAD

Definiciones Auxiliares de SimCity

avanzarValido : SimCity s × dicc(Pos × Construcción) cs ⟶ boolean
avanzarValido(s, cs) ≡ ¬ vacio?(claves(cs)) ∧
(∀ p : Pos) (def?(p, cs) ⇒$_{\text{L}}$
(¬ p ∈ claves(construcciones(s)) ∧
¬ esRio(p, mapa(s)) ∧
(obtener(p, cs) = "*casa*" ∨ obtener(p, cs) = "*comercio*"))
)

unirValido : Simcity a × SimCity b ⟶ boolean
unirValido(a, b) ≡ (∀ p : Pos)(def?(p, construcciones(a)) ⇒$_{\text{L}}$
¬ esRio(p, mapa(b)) ∧
(∄ $otra$: Pos)[1](def?($otra$, construcciones(a)) ∧$_{\text{L}}$
obtener($otra$, construcciones(a)) > obtener(p, construcciones(a))
) ⇒$_{\text{L}}$ ¬ def?(p, construcciones(b))
) ∧
(∀ p : Pos)(def?(p, construcciones(b)) ⇒$_{\text{L}}$
¬ esRio(p, mapa(a)) ∧
(∄ $otra$: Pos)[1](def?($otra$, construcciones(b)) ∧$_{\text{L}}$
obtener($otra$, construcciones(b)) > obtener(p, construcciones(b))
) ⇒$_{\text{L}}$ ¬ def?(p, construcciones(a))
)

1. Si en la posición hay una construcción de nivel máximo, no puede colisionar con una construcción del otro SimCity.

1.4. Servidor

TAD SERVIDOR

géneros Server

exporta Server, observadores, generadores, verMapa, verCasas, verComercios, verPopularidad, agregarCasa, agregarComercio y verTurno

usa SimCity, Mapa, Nombre, Pos, Construcción, Nivel, Nat, Bool, dicc($\alpha \times \beta$), conj(α)

igualdad observacional

$$(\forall s, s' : \text{Server}) \left(s =_{\text{obs}} s' \iff \begin{pmatrix} \text{partidas}(s) =_{\text{obs}} \text{partidas}(s') \wedge_{\text{L}} \\ \text{congeladas}(s) =_{\text{obs}} \text{congeladas}(s') \end{pmatrix} \right)$$

observadores básicos

partidas	: Server	$\longrightarrow$	dicc(Nombre $\times$ SimCity)
congeladas	: Server	$\longrightarrow$	conj(Nombre)

generadores

nuevoServer	:	$\longrightarrow$	Server	
nuevaPartida	: Server s $\times$ Nombre p $\times$ Mapa	$\longrightarrow$	Server	$\{\neg$ def?(p, partidas(s))$\}$
unirPartidas	: Server s $\times$ Nombre $p1$ $\times$ Nombre $p2$	$\longrightarrow$	Server	$\{$unionValida(s, $p1$, $p2$)[1]$\}$
avanzarTurnoPartida	: Server s $\times$ Nombre p $\times$ dicc(Pos$\times$Construcción) cs	$\longrightarrow$	Server	$\{$avanzarValido(s, p)[1]$\}$

otras operaciones

verMapa	: Server s $\times$ Nombre p	$\longrightarrow$	Mapa	$\{$def?(p, partidas(s))$\}$
verCasas	: Server s $\times$ Nombre p	$\longrightarrow$	dicc(Pos $\times$ Nivel)	$\{$def?(p, partidas(s))$\}$
verComercios	: Server s $\times$ Nombre p	$\longrightarrow$	dicc(Pos $\times$ Nivel)	$\{$def?(p, partidas(s))$\}$
verPopularidad	: Server s $\times$ Nombre p	$\longrightarrow$	Nat	$\{$def?(p, partidas(s))$\}$
verTurno	: Server s $\times$ Nombre p	$\longrightarrow$	Nat	$\{$def?(p, partidas(s))$\}$
agregarCasa	: Server s $\times$ Nombre p $\times$ Pos pos	$\longrightarrow$	Server	$\{$agregarValido(s, p, pos)[1]$\}$
agregarComercio	: Server s $\times$ Nombre p $\times$ Pos pos	$\longrightarrow$	Server	$\{$agregarValido(s, p, pos)[1]$\}$

axiomas $\forall s$: Server, $\forall p,\ p',\ p''$: Nombre, $\forall m$: Mapa, $\forall pos$: Pos

partidas(nuevoServer)	$\equiv$	vacio
partidas(nuevaPartida(s, p, m))	$\equiv$	definir(p, iniciar(m), partidas(s))
partidas(unirPartidas(s, p, p'))	$\equiv$	definir(p, unir(obtener(p, partidas(s)), obtener(p', partidas(s))), partidas(s))
partidas(avanzarTurnoPartida(s, p, cs))	$\equiv$	definir(p, avanzarTurno(obtener(p, partidas(s)), cs), partidas(s))
congeladas(nuevoServer)	$\equiv$	$\emptyset$
congeladas(nuevaPartida(s, p, m))	$\equiv$	congeladas(s)
congeladas(unirPartidas(s, p, p'))	$\equiv$	Ag(p', congeladas(s))
congeladas(avanzarTurnoPartida(s, p))	$\equiv$	congeladas(s)

1. definido en el apartado Definiciones Auxiliares de Servidor.

otros ax. $\forall s$: Server, $\forall p$: Nombre, $\forall pos$: Pos

verMapa(s, p) $\equiv$ mapa(obtener(p, partidas(s)))
verCasas(s, p) $\equiv$ casas(obtener(p, partidas(s)))
verComercios(s, p) $\equiv$ comercios(obtener(p, partidas(s)))
verPopularidad(s, p) $\equiv$ popularidad(obtener(p, partidas(s)))
verTurno(s, p) $\equiv$ turnos(obtener(p, partidas(s)))
agregarCasa(s, p, pos) $\equiv$ avanzarTurnoPartida(s, p, definir(pos, "*casa*", vacio))
agregarComercio(s, p, pos) $\equiv$ avanzarTurnoPartida(s, p, definir(pos, "*comercio*", vacio))

Fin TAD

Definiciones Auxiliares de Servidor

unionValida : Server s × Nombre p × Nombre p' $\longrightarrow$ boolean
unionValida(s, p, p') $\equiv$ def?(p, partidas(s)) $\wedge$ def?(p', partidas(s)) $\wedge$ $p \notin$ congeladas(s) $\wedge_{\text{L}}$
unirValido(obtener(p, partidas(s)), obtener(p', partidas(s)))[1]

avanzarValido : Server s × Nombre p × dicc(Pos×Construcción) cs $\longrightarrow$ boolean
avanzarValido(s, p, cs) $\equiv$ def?(p, partidas(s)) $\wedge$
$p \notin$ congeladas(s) $\wedge$
$\neg$ vacia?(claves(cs)) $\wedge_{\text{L}}$
($\forall$ pos: Pos)($pos \in$ claves(cs) $\Rightarrow_{\text{L}}$
obtener(pos, cs) $\in$ {"*casa*", "*comercio*"} $\wedge$
$\neg$ esRio(pos, verMapa(s, p)) $\wedge$
$\neg$ def?(pos, verCasas(s, p)) $\wedge$
$\neg$ def?(pos, verComercios(s, p))
)

agregarValido : Server s × Nombre p × Pos pos $\longrightarrow$ boolean
agregarValido(s, p, pos) $\equiv$ def?(p, partidas(s)) $\wedge$ $p \notin$ congeladas(s) $\wedge_{\text{L}}$
$\neg$ def?(pos, verCasas(s, p)) $\wedge$ $\neg$ def?(pos, verComercios(s, p)) $\wedge$
$\neg$ esRio(pos, verMapa(s, p))

1. definido en el apartado Definiciones Auxiliares de SimCity.

2. Diseño

2.1. Aliases

Pos es tupla<x: Nat × y: Nat>

Construcción es String

Nombre es String

Nivel es Nat

2.2. Módulo Mapa

2.2.1. Interfaz

Interfaz

usa: Conjunto Lineal, Nat, Bool, Pos

exporta: todo

se explica con: Mapa

géneros: `Mapa`

Operaciones básicas de Mapa

horizontales(**in** m : `Mapa`) → res : `Conj(Nat)`
Pre ≡ {true}
Post ≡ {$r\hat{e}s =_{\text{obs}}$ horizontales($\hat{m}$)}
Complejidad: **O**(#horizontales($\hat{m}$))
Descripción: Devuelve el conjunto de ríos horizontales.
Aliasing: Por copia.

verticales(**in** m : `Mapa`) → res : `Conj(Nat)`
Pre ≡ {true}
Post ≡ {$r\hat{e}s =_{\text{obs}}$ verticales($\hat{m}$)}
Complejidad: **O**(#verticales($\hat{m}$))
Descripción: Devuelve el conjunto de ríos verticales.
Aliasing: Por copia.

crear(**in** hs : `conj(Nat)` , **in** vs : `conj(Nat)`) → res : `Mapa`
Pre ≡ {true}
Post ≡ {$r\hat{e}s =_{\text{obs}}$ crear($\hat{hs}$, $\hat{vs}$)}
Complejidad: **O**(copy(hs) + copy(vs))
Descripción: Crea un mapa.
Aliasing: Por copia.

esRio(**in** m : `Mapa`, **in** p : `Pos`) → res : `Bool`
Pre ≡ {true}
Post ≡ {$r\hat{e}s =_{\text{obs}}$ esRio($\hat{p}$, $\hat{m}$)}
Complejidad: **O**(#horizontales(m) + #verticales(m))
Descripción: Verifica si en determinada pos hay un río.

suma(**in** $m1$: `Mapa`, **in** $m2$: `Mapa`) → res : `Mapa`
Pre ≡ {true}
Post ≡ {$r\hat{e}s =_{\text{obs}}$ $\hat{m1}$ + $\hat{m2}$}
Complejidad: **O**(copy($m1$) + copy($m2$))
Descripción: Une dos Mapas.
Aliasing: Por copia.

2.2.2. Representación

Representación

Representación de mapa

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

`Mapa` **se representa con** `estr`

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

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

Abs : estr e $\longrightarrow$ Mapa {Rep(e)}
Abs(e) $\equiv$ m: Mapa | horizontales(m) $=_{\text{obs}}$ e.horizontales $\wedge$ verticales(m) $=_{\text{obs}}$ e.verticales

2.2.3. Implementación

Algoritmos

---

crear(**in** hs : conj(Nat), **in** vs : conj(Nat)) $\longrightarrow$ res : estr
1: res.horizontales $\leftarrow$ hs
2: res.verticales $\leftarrow$ vs
3: **return** res
Complejidad: O(copy(hs) + copy(vs))

Suma(**in** m1: mapa, **in** m2: mapa) $\longrightarrow$ res : mapa
1: mapa res $\leftarrow$ copiar(m1)

2: itConj(Nat) itH $\leftarrow$ crearIt(horizontales(m2))
3: **while**(haySiguiente(itH)) :
4: Ag(horizontales(res), siguiente(itH))
5: avanzar(itH)

6: itConj(Nat) itV $\leftarrow$ crearIt(verticales(m2))
7: **while**(haySiguiente(itV)) :
8: Ag(verticales(res), siguiente(itV))
9: avanzar(itV)
10: **return** res
Complejidad: O(copiar(m1) + copiar(m2))

---

esRio(**in** p: Pos, **in** m: Mapa) $\longrightarrow$ res : Bool
1: **return** p.x $\in$ verticales(m) $\vee$ p.y $\in$ horizontales(m)
Complejidad: O(#horizontales(m) + #verticales(m))

---

verticales(**in** m: mapa) $\longrightarrow$ res : Conj(Nat)
1: **return** m.verticales
Complejidad: O(copy(m.verticales))

---

horizontales(**in** m: mapa) $\longrightarrow$ res : Conj(Nat)
1: **return** m.horizontales
Complejidad: O(copy(m.horizontales))

2.3. Módulo SimCity

2.3.1. Interfaz

Interfaz

usa: Mapa, Diccionario Lineal, Pos, Nivel, Nat

exporta: todo

se explica con: SimCity

géneros: `SimCity`

Operaciones básicas de SimCity

Sea S: SimCity, N = popularidad(S), $\{u_0 \ ... \ u_N\} = U$: el conjunto de SimCities en union con S[1] y S, y $construcciones = \sum_{i=0}^{N}(\#(construcciones_i))$, donde $construcciones_i$ es el conjunto de casas y comercios definidos en u_i por medio de $AvanzarTurno$.[2]

MAPA(**in** S : `SimCity`) $\rightarrow$ res : `Mapa`
Pre $\equiv$ {true}
Post $\equiv$ $\{r\hat{e}s =_{\text{obs}} \text{mapa}(\hat{S})\}$
Complejidad: $\mathbf{O}(\sum_{i=0}^{N}(mapa_i))$, donde $mapa_i$ es el Mapa original[3] de u_i.
Descripción: Retorna el mapa sobre el que se desarrolla el juego actual.
Aliasing: No. Genera una copia.

CASAS(**in** S : `SimCity`) $\rightarrow$ res : `DiccLineal(Pos, Nivel)`
Pre $\equiv$ {true}
Post $\equiv$ $\{r\hat{e}s =_{\text{obs}} \text{casas}(\hat{S})\}$
Complejidad: $\mathbf{O}(construcciones^2) + \mathbf{O}(N)$
justificación: Una implementación de SimCity debe poder, en el peor caso, retornar una copia de todas sus casas en tiempo lineal sobre el conjunto que las representa. Pero, dado que es esperable que un SimCity, compuesto por un conjunto de uniones, sea representado como un árbol, podemos suponer que para resolver colisiones del tipo 'se queda el primero', cada nueva casa a copiar debe primero evaluar si no pertenece ya a los niveles superiores, incluyendo también a los comercios. Resultando en un peor caso, holgado, de O($construcciones^2$), donde no hay colisiones. Dado que un SimCity puede no tener construcciones si está recién iniciado, se debe considerar que el peor caso también puede estar dado por la cantidad de nodos a recorrer en el árbol. Es decir O(N). Por ejemplo, si se anidan 100 SimCities recién iniciados.
Descripción: Retorna las posiciones y respectivos niveles de todas las casas en el juego actual.
Aliasing: No. Genera una copia.

COMERCIOS(**in** S : `SimCity`) $\rightarrow$ res : `DiccLineal(Pos, Nivel)`
Pre $\equiv$ {true}
Post $\equiv$ $\{r\hat{e}s =_{\text{obs}} \text{comercios}(\hat{S})\}$
Complejidad: $\mathbf{O}(construcciones^2) + \mathbf{O}(N)$
justificación: similiar a $casas(s)$.
Descripción: Retorna las posiciones y respectivos niveles de todos los comercios en el juego actual.
Aliasing: No. Genera una copia.

POPULARIDAD(**in** S : `SimCity`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{r\hat{e}s =_{\text{obs}} \text{popularidad}(\hat{S})\}$
Complejidad: $\mathbf{O}(1)$
Descripción: Retorna la cantidad total de uniones que se realizaron para conformar la partida actual.
Aliasing: No. Por copia.

1. Este conjunto incluye también a los SimCities provenientes de las uniones propias a cada SimCity en unión directa con S.
2. En particular, notar que $construcciones_i \geq \#(\text{claves}(\text{construcciones}(\hat{S})))$, por posibles colisiones permitidas entre los u_i. Dado la necesidad de resolver la union en $\mathbf{O}(1)$, no se puede mantener un registro de construcciones sin repetidos. De éste modo, se contempla el total real de construcciones definidas al momento de calcular la complejidad que tendrán las operaciones.
3. Es decir, aquel con el que se inició originalmente el simCity.

TURNOS(**in** S : `SimCity`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {true}
Post $\equiv$ {$r\hat{e}s$ $=_{\text{obs}}$ turnos($\hat{S}$)}
Complejidad: **O**(1)
Descripción: Retorna la cantidad de turnos que pasaron desde que se inició el SimCity.
Aliasing: No. Genera una copia.

INICIAR(**in** m : `Mapa`) $\rightarrow$ res : `SimCity`
Pre $\equiv$ {true}
Post $\equiv$ {$r\hat{e}s$ $=_{\text{obs}}$ iniciar($\hat{m}$)}
Complejidad: **O**(copy(m))
Descripción: Crea un nuevo SimCity.
Aliasing: No. Genera una copia.

AVANZARTURNO(**in/out** S : `SimCity`, **in** cs : `diccLineal(Pos, Construccion)`)
Pre $\equiv$ {avanzarValido[1]($\hat{s}$, $\hat{cs}$) $\wedge$ $\hat{S}$ = S_0}
Post $\equiv$ {$\hat{S}$ $=_{\text{obs}}$ avanzarTurno(S_0, $\hat{cs}$)}
Complejidad: **O**(1)
Descripción: Avanza el turno de un SimCity.
Aliasing: Genera una copia de las posiciones en el diccionario.

UNIR(**in/out** $S1$: `SimCity`, **in** $S2$: `SimCity`)
Pre $\equiv$ {unionValida[1]($\hat{S1}$, $\hat{S2}$) $\wedge$ $\hat{S1}$ = S_0}
Post $\equiv$ {$\hat{S1}$ $=_{\text{obs}}$ unir(S_0, $\hat{S2}$)}
Complejidad: **O**(1)
Descripción: Une dos SimCities.
Aliasing: Se guarda una referencia a S2 en S1. Cualquier cambio sobre S2 modificará a S1.

1. definido en Definiciones Auxiliares de SimCity.

2.3.2. Representación

Representación

Un *SimCity* se compone por la *ubicacion* y *nivel* de una serie de *construcciones*, de tipo *Casa* o *Comercio*, sobre un *Mapa*, y de una *Popularidad* respecto a la cantidad de uniones que lo modificaron.

reescribir La ubicación de las casas se representan sobre un diccionario lineal con clave *Pos* y significado *Nivel*. La ubicación de los comercios se representan similarmente, pero su significado responde a un *NivelBase* de tipo *Nat* a partir del cual se calcula propiamente su *Nivel*. El mapa es de tipo *Mapa* y las uniones se representan a través de una *lista* de *Hijos* que contiene punteros a los *SimCities* unidos e información relevante para calcular el nivel de sus construcciones. Ya que, una vez unido a otro, un *SimCity* debe permanecer sin modificar.

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

donde `estr` es `tupla`(*antiguedad*: `Nat` ,
turno : `Nat`,
popularidad : `Nat`,
mapa : `Mapa`,
construcciones : `lista(puntero(diccLineal(Pos, Construcción)))` ,
uniones : `lista(hijo)`)

donde `hijo` es `tupla`(*sc* : `puntero(estr)` ,
turnoUnido : `Nat`)

donde `pos` es `tupla`(x : `Nat` , y : `Nat`)

$$
\begin{aligned}
&\text{Rep} : \text{estr}^1 \longrightarrow \text{boolean} \\
&\text{Rep}(e) \equiv \text{true} \iff (\\
&\quad (\forall\, h : \text{hijo})(\text{esta?}(h,\ e.\mathit{uniones}) \Rightarrow_{\text{L}} \\
&\qquad (h.\mathit{sc} \neq \text{NULL} \wedge_{\text{L}} \mathit{rep}(*(h.\mathit{sc})))^2 \wedge_{\text{L}} \\
&\qquad (e.\mathit{turno} \geq h.\mathit{turnoUnido})^3 \wedge \\
&\qquad (\forall\, h' : \text{hijo})(\text{esta?}(h',\ e.\mathit{uniones}) \wedge_{\text{L}} \text{pos}(h',\ e.\mathit{uniones}) > \text{pos}(h,\ e.\mathit{uniones}) \Rightarrow_{\text{L}} \\
&\qquad\quad h'.\mathit{turnoUnido} \leq h.\mathit{turnoUnido} \\
&\qquad)^4 \\
&\quad) \wedge_{\text{L}} \\
&\quad (\forall\, p : \text{puntero(dicc(Pos, Construccion))})(\text{esta?}(p,\ e.\mathit{construcciones}) \Rightarrow_{\text{L}} \\
&\qquad (\nexists\, p' : \text{puntero(dicc(Pos, Construccion))})(\text{esta?}(p',\ e.\mathit{construcciones}) \wedge_{\text{L}} \\
&\qquad\quad \text{pos}(p,\ e.\mathit{construcciones}) \neq \text{pos}(p'\, e.\mathit{construcciones}) \wedge_{\text{L}} \\
&\qquad\quad \text{claves}(*(p)) \cap \text{claves}(*(p')) \neq \emptyset \\
&\qquad) \\
&\quad)^5 \wedge_{\text{L}} \\
&\quad (\text{long}(e.\mathit{construcciones}) = e.\mathit{turno})^6 \wedge_{\text{L}} \\
&\quad (\&e \notin \text{Unidos})^7 \wedge_{\text{L}} \\
&\quad (\forall\, p : \text{puntero(estr)})(p \in \text{Unidos} \Rightarrow_{\text{L}} \\
&\qquad e.\mathit{antiguedad} \geq (*p).\mathit{antiguedad} \\
&\quad)^8 \wedge \\
&\quad (e.\mathit{popularidad} = \#(\text{Unidos}))^9 \wedge \\
&\quad (\forall\, p : \text{Pos})(p \in \text{Casas} \Rightarrow_{\text{L}} \\
&\qquad \neg\text{esRio}(p, \text{Mapas}) \wedge (\text{nivel}(e,\ p) \leq e.\mathit{antiguedad}) \\
&\quad)^{10} \wedge \\
&\quad (\forall\, p : \text{Pos})(p \in \text{Comercios} \Rightarrow_{\text{L}} \\
&\qquad \neg\text{esRio}(p, \text{Mapas}) \wedge (\text{nivel}(e,\ p) \leq e.\mathit{antiguedad}) \\
&\quad)^{11} \wedge \\
&\quad (\nexists\, p : \text{Pos})((p \in \mathit{Comercios} \vee p \in \mathit{Casas}) \wedge_{\text{L}} \mathit{nivel}(e,\ p) = e.\mathit{antiguedad})^{12} \wedge \\
&\quad \text{unionesValidas}(e,\ e.\mathit{uniones})^{13} \\
&\quad)
\end{aligned}
$$

donde

$$
\begin{aligned}
\text{Unidos} &\equiv \text{unirPunteros}(e.\mathit{uniones}) \\
\text{Casas} &\equiv \text{unirCasas}(\text{Ag}(\&e, \text{Unidos}))
\end{aligned}
$$

Comercios $\equiv$ unirComercios(Ag(&e, Unidos))
Mapas $\equiv$ unirMapas(Ag(&e, Unidos))

Abs : estr e $\longrightarrow$ SimCity {Rep(e)}
Abs(e) $\equiv$ sc : SimCity |
mapa(sc) $=_{\text{obs}}$ Mapas $\wedge$
casas(sc) $=_{\text{obs}}$ nivelar(Casas) $\wedge$
comercios(sc) $=_{\text{obs}}$ nivelar(Comercios) $\wedge$
popularidad(sc) $=_{\text{obs}}$ $e.popularidad$

donde
Unidos $\equiv$ unirPunteros($e.uniones$)
Casas $\equiv$ unirCasas(Ag(&e, Unidos))
Comercios $\equiv$ unirComercios(Ag(&e, Unidos))
Mapas $\equiv$ unirMapas(Ag(&e, Unidos))

Definiciones Auxiliares

Los siguientes reemplazos sintácticos están confinados al contexto del invariante de representación y la función de abstracción del SimCity. En éste sentido, se considera restricción implícita, para cada uno, ser evaluado en un estado que satisfaga parcialmente la representación,-en términos de lógica ternaria-, al momento de 'llamada' dentro de la misma.

unirPunteros : secu(hijo) s $\longrightarrow$ conj(puntero(estr)) {$(\forall h : \text{hijo})(h \in s \Rightarrow_{\text{L}} h.sc \neq \text{NULL})$}
unirPunteros(s) $\equiv$ _unirPunteros(s, $\emptyset$)

_unirPunteros : secu(hijo) s $\times$ conj(puntero(estr)) ps $\longrightarrow$ conj(puntero(estr)) {eq. unirPunteros}
_unirPunteros(s, ps) $\equiv$ **if** vacia?(s) **then**
ps
else if prim(s).sc $\in$ ps **then** // por si hay loops
_unirPunteros(fin(s), ps)
else
_unirPunteros((*(prim(s).sc)).$uniones$, Ag(prim(s).sc, ps)) $\cup$
_unirPunteros(fin(s), Ag(prim(s).sc, ps))
// al unir se descarta el duplicado
fi

unirMapas : conj(puntero(estr)) ps $\longrightarrow$ Mapa
{$(\forall p : \text{puntero(estr)})(p \in ps \Rightarrow_{\text{L}} (p \neq \text{NULL} \wedge_{\text{L}} \text{rep(*p)}))$}
unirMapas(ps) $\equiv$ **if** vacio?(ps) **then**
crear($\emptyset$, $\emptyset$)
else
(*(dameUno(ps))).$mapa$ + UnirMapas(sinUno(ps))
fi

1. Se asume el traspaso de toda estructura de representación a su equivalente abstracto (se aplica el sombrerito).
2. Cada hijo apunta a un Simcity válido.
3. El turno es mayor o igual al turno de cualquier unión inmediata.
4. Las uniones están ordenadas de más antiguas a más recientes.
5. Las construcciones en éste nivel no se solapan.
6. Se agregó al menos un conjunto de construcciones por cada turno interno.
7. La estructura no loopea consigo misma.
8. La antiguedad es mayor o igual a la antiguedad de cualquier simCity hijo.
9. La popularidad es igual a la cantidad de uniones total.
10. Ninguna casa está sobre un río perteneciente a cualquier mapa en la unión y su nivel es menor o igual a la antiguedad de la partida.
11. Ninguna construcción está sobre un río perteneciente a cualquier mapa en la unión y su nivel es menor o igual a la antiguedad de la partida.
12. Existe almenos una casa o comercio cuyo nivel es máximo, en el sentido que corresponde a la antiguedad de la partida.
13. No se solapan posiciones máximas entre esta estructura hasta el hijo 'x', y ese hijo, para todo hijo. Notar que no importan los turnos posteriores al momento de union, porque cualquier construcción agregada posteriormente no podrá ser máxima o modificar los máximos al momento de la unión. Dadas las condiciones ya establecidas hasta este momento.

unirCasas : conj(puntero(estr)) ps $\longrightarrow$ dicc(Pos $\times$ Nat) {eq. unirMapas}
unirCasas(ps) $\equiv$ **if** vacio?(ps) **then**
vacio
else
$it((*(dameUno(ps))).construcciones,$ "$casa$") $\cup_{dicc}$ unirCasas(sinUno(ps))
fi

unirComercios : conj(puntero(estr)) ps $\longrightarrow$ dicc(Pos $\times$ Nat) {eq. unirMapas}
unirComercios(ps) $\equiv$ **if** vacio?(ps) **then**
vacio
else
$it((*(dameUno(ps))).construcciones,$ "$comercio$") $\cup_{dicc}$ unirComercios(sinUno(ps))
fi

it : secu(puntero(dicc(Pos $\times$ Construccion))) s $\times$ Construccion c $\longrightarrow$ dicc(Pos $\times$ Nat)
// el significado representa el turno en que se agregó la construcción.
it(s, c) $\equiv$ _it(s, c, 0)

_it : secu(puntero(dicc(Pos $\times$ Construccion))) s $\times$ Construccion c $\times$ Nat n $\longrightarrow$ dicc(Pos $\times$ Nat)
_it(s, c, n) $\equiv$ **if** vacia?(s) **then** vacio **else** filtrar(*(prim(s)), c, n) $\cup_{dicc}$ it(fin(s), c, n + 1) **fi**

filtrar : dicc(Pos $\times$ Construccion) $\times$ Construccion $\times$ Nat $\longrightarrow$ dicc(Pos $\times$ Nat)
filtrar(d, c, n) $\equiv$ **if** vacio?(d) **then**
vacio
else if obtener(clave, d) = c **then**
definir(clave, n, filtrar(borrar(clave, d), c, n))
else
filtrar(borrar(clave, d), c, n)
fi
donde
clave $\equiv$ dameUno(claves(d))

pos : secu(α) s $\times$ α a $\longrightarrow$ Nat {esta?(a, s)}
pos(s, a) $\equiv$ **if** prim(s) = a **then** 0 **else** 1 + pos(fin(s), a) **fi**

unionesValidas : estr e $\times$ secu(hijo) s $\longrightarrow$ bool
{$s \subseteq e.uniones \wedge_{\text{L}} (\forall h : \text{hijo})(h \in s \Rightarrow_{\text{L}} h.sc \neq \text{NULL} \wedge_{\text{L}} \text{rep}(*(h.sc)))$}
unionesValidas(e, s) $\equiv$ vacio?(s) $\vee_{\text{L}}$ (maxcons(e, izq) $\cap$ maxcons(e, der) $=$ $\emptyset$ $\wedge$ unionesValidas(e, com(s)))
donde
com $\equiv$ unirPunteros(com(s))
ult $\equiv$ ult(s) $\bullet$ <>
casascom $\equiv$ unirCasas(com)
comercom $\equiv$ unirComercios(com)
casasult $\equiv$ unirCasas(ult)
comerult $\equiv$ unirComercios(ult)
izq $\equiv$ claves(casascom) $\cup$ claves(comercom)
der $\equiv$ claves(casasult) $\cup$ claves(comerult)

maxcons : estr e $\times$ conj(Pos) c $\longrightarrow$ conj(Pos) {$(\forall p : \text{Pos})(p \in c \Rightarrow_{\text{L}} p \in \text{posiciones}(e))$}
maxcons(e, c) $\equiv$ _maxcons(e, c, $\emptyset$, 0)

_maxcons : estr e × conj(Pos) c × conj(Pos) max × Nat n ⟶ conj(Pos) {eq. maxcons}
_maxcons(e, c, max, n) ≡ **if** vacio?(c) **then**
max
else if nivel$_i$ > n **then**
_maxcons(e, sinUno(c), Ag(pos$_i$, ∅), nivel$_i$)
else if nivel$_i$ = n **then**
_maxcons(e, sinUno(c), Ag(pos$_i$, max), n)
else
_maxcons(e, sinUno(c), max, n)
fi
donde
pos$_i$ ≡ dameUno(c)
nivel$_i$ ≡ nivel(e, pos$_i$)

nivel : estr e × Pos pos ⟶ Nat {p ∈ posiciones(e)}
nivel(e, pos) ≡ **if** def?(pos, Casas) **then**
nivelesPorUnion(e, pos)
else
max(nivelesPorUnion(e, pos), nManhattan)
fi
donde:
Unidos ≡ unirPunteros($e.uniones$)
Casas ≡ unirCasas(Ag(&e, Unidos))
Comercios ≡ unirComercios(Ag(&e, Unidos))
nManhattan ≡ manhattan(e, pos, Casas)

nivelesPorUnion : estr e × Pos pos ⟶ Nat {eq. nivel}
nivelesPorUnion(e, pos) ≡ **if** def?(pos, Construcciones) **then**
$e.turno$ − obtener(pos, Construcciones)
else
$e.turno$ − hijoCorrecto.$turnoUnion$ + nivelesPorUnion(hijoCorrecto.sc, pos)
fi
donde:
Construcciones ≡ it($e.construcciones$, "$casa$") $\cup_{dicc}$ it($e.construcciones$, "$comercio$")
hijoCorrecto ≡ llegar($e.uniones$, pos)

llegar : secu(hijo) s × Pos p ⟶ hijo {(∀ h : hijo)(esta?(h, s) ⇒$_L$ $h.sc$ ≠ NULL ∧$_L$ rep(∗($h.sc$))) ∧$_L$
(∃h : hijo)(esta?(h, s) ∧$_L$ p ∈ $posiciones(h.sc)$)}
llegar(s, p) ≡ **if** def?(pos, unirCasas($hijo_i$)) ∨ def?(pos, unirComercios($hijo_i$)) **then**
prim(s)
else
llegar(fin(s))
fi
donde
hijo$_i$ ≡ Ag(prim(s).sc, ∅)

manhattan : estr e × Pos p × Dicc(Pos × Nat) d ⟶ Conj(Pos)
{(∀p' :Pos)(p' ∈ claves(d) ⇒$_L$ p' ∈ $posiciones(e)$)}
manhattan(e, p, d) ≡ **if** vacio?(claves(d)) **then**
0
else if distManhattan(p, proximo) ≤ 3 ∧ p ≠ proximo **then**
max(nivel(e, proximo)), manhattan(p, borrar(proximo, d))
else
manhattan(e, p, borrar(proximo, d))
fi
donde
proximo ≡ dameUno(claves(d))

posiciones : estr e $\longrightarrow$ dicc(Pos $\times$ Nat)
$\{(\forall h : \text{hijo})(h \in e.uniones \Rightarrow_{\text{L}} h.sc \neq \text{NULL} \wedge_{\text{L}} \text{rep}(*(h.sc)))\}$
posiciones(e) $\equiv$ claves(unirCasas(Ag(&e, unirPunteros($e.uniones$)))) $\cup$
claves(unirComercios(Ag(&e, unirPunteros($e.uniones$))))

nivelar : estr e $\times$ dicc(Pos $\times$ Nat) d $\longrightarrow$ dicc(Pos $\times$ Nat)
$\{(\forall p : \text{Pos})(p \in \text{claves}(d) \Rightarrow_{\text{L}} p \in posiciones(e))\}$
nivelar(e, d) $\equiv$ **if** vacio?(d) **then** vacio **else** definir(clave, nivel(e, clave), nivelar(e, borrar(clave, d))) **fi**
donde
clave $\equiv$ dameUno(claves(d))

2.3.3. Implementación

Algoritmos

Interpretamos que el **inout** se refiere a que determinado parametro esta siendo modificado y se toma por contexto si pasa por referencia.

Sea U $\equiv \{u_1, u_2, \ldots, u_n\}$
tal que cada uno de esos u_i son los simcities que componen al simcity original
llamamos nodos : #U
llamamos sumConstrucciones : $\sum_{i=1}^{nodos}$(copiar(u_i.construcciones))
llamamos sumMapas : $\sum_{i=1}^{nodos}$(u_i.mapa)

---

iniciar(**in** m: Mapa) $\longrightarrow$ res : res
1: res.turno $\leftarrow$ 0
2: res.antiguedad $\leftarrow$ 0
3: res.popularidad $\leftarrow$ 0
4: res.mapa $\leftarrow$ m
5: res.construcciones $\leftarrow$ vacio()
6: res.uniones $\leftarrow$ vacia()
7: **return** res
Complejidad: O(copy(m))

---

avanzarTurno(**inout** SimCity s, **in** dicc(Pos, Construccion) cs)
1: s.turno $\leftarrow$ s.turno + 1
2: s.antiguedad $\leftarrow$ s.antiguedad + 1
3: s.construcciones.agregarAtras(direccion(cs))
Complejidad: O(1)

---

unir(**inout** SimCity s1, **in** Simcity s2)
1: s1.popularidad $\leftarrow$ s1.popularidad + s2.popularidad + 1
2: s1.antiguedad $\leftarrow$ max(s1.antiguedad, s2.antiguedad)
3: hijo nuevoHijo $\leftarrow$ <direccion(s2), s1.turno>
4: agregarAtras(s1.uniones, nuevoHijo)
Complejidad: O(1)

---

mapa(**in** SimCity s) $\rightarrow$ res : Mapa
1: Mapa res $\leftarrow$ s.mapa
2: **for**(nat i $\leftarrow$ 0; i < long(s.uniones); i $\leftarrow$ i + 1) :
3: res $\leftarrow$ res + **mapa**(*s.uniones[i].sc)
4: **return** res
Complejidad: O(sumMapas)
Cabe aclarar que la suma de los mapas esta definida.

---

listDeTipo(**in** SimCity s, **in** construccion tipo) → res : dicc(Pos, Nivel)
1: dicc(pos, nivel) res ← vacio()
2: **for**(nat i ← 0; i < long(s.construcciones); i ← i + 1) :
3: itDicc(Pos, Construccion) itCs ← crearIt(*s.construcciones[i]);
4: **while**(haySiguiente(itCs)) :
5: Pos p ← siguienteClave(itCs)
6: construccion c ← siguienteSignificado(itCs)
7: **if**(c == tipo) :
8: definirRapido(res, p, s.turno - (i+1))
9: avanzar(itCs)
Complejidad: $O(\sum_{i=1}^{long(s.construcciones)}(\#claves(*s.construcciones[i])))$
Justificación: (2) recorre cada uno de los diccionarios definidos en s.construcciones y para cada uno recorre todas las definiciones del mismo, (4). Por lo tanto la **complejidad** del algoritmo es la suma de todas las construcciones definidas en ese SimCity.

---

casas(**in** SimCity s) → res : dicc(Pos, Nivel)
1: dicc(pos, nivel) res ← copiar(listDeTipo(s, "casa"))
2: dicc(pos, nivel) comerciosTotales ← copiar(listDeTipo(s, "comercio"))
3: **for**(nat i ← 0; i < long(s.uniones); i ← i + 1) :
4: itDicc(Pos, Nivel) itCs ← crearIt(**casas**(*s.uniones[i].sc))
5: **while**(haySiguiente(itCs)) :
6: Pos p ← siguienteClave(itCs)
7: Nivel n ← siguienteSignificado(itCs)
8: **if**(¬def?(res, p) ∧ ¬def?(comerciosTotales, p)) :
9: definirRapido(res, p, s.turno - s.uniones[i].turnoUnido + n)
10: avanzar(itCs)
11: comerciosTotales ← comerciosTotales ∪ listDeTipo(*s.uniones[i].sc, "comercio")
12: **return** res
Complejidad: O(sumConstrucciones^2 * nodos + nodos)
Justificación:
Sabemos que este algoritmo va a recorrer por cada SimCity que pertenece al SimCity inicial 1 vez. Ya que el paso recursivo (4) llama a un simcity como hijo de otro simcity y cada simcity como maximo es hijo de 1 solo otro simcity
–> nodos

En cada una de estas recursiones, primero (1) y (2) van a generar un diccionario con las casas y los comercios de ese simcity particular.
Si repetimos este procedimiento para cada uno de los Simcities que pertenecen al simcity original vamos a estar recorriendo 1 vez las construcciones de cada simcity por lo tanto la suma de recorrer las construcciones de cada uno es igual a sumConstrucciones.
–> sumConstrucciones

(3) = #(hijos de simCity particular)
(5) = #(casas totales de un hijo particular)
(8) = #(comerciosTotales) + #(casasTotales) = sumConstrucciones
(3) * (5) * (8) = (casasTotales) * sumConstrucciones ≤ sumConstrucciones^2
por lo tanto si hace en el peor caso sumConstrucciones^2 para cada Simcity particular
–> sumConstrucciones^2 * nodos

luego en (11) hace la union de los comercios que ya analizo con los comercios del simcity que acaba de analizar pero esta accion la hace 1 sola vez por simcity. Pero como la union es por copia y no por referencia esto es sumConstrucciones^2 para cada simcity particular.
–> sumConstrucciones^2 * nodos

finalmente: nodos + sumConstrucciones + sumConstrucciones^2 * nodos + sumConstrucciones^2 * nodos
–> sumConstrucciones^2 * nodos + nodos

---

comercios(**in** SimCity s) → res : dicc(Pos, Nivel)
1: dicc(Pos, Nivel) casasTotales ← casas(s)
2: dicc(Pos, Nivel) comerciosTotales ← comerciosAux(s, casasTotales)
2: **return** manhatizar(comerciosTotales, casasTotales)
Complejidad: O(manhatizar(comerciosAux(s, casasTotales), casasTotales) + casas(s))

comerciosAux(**inout** SimCity s, **inout** casasTotales) → res : dicc(Pos, Nivel)
1: dicc(pos, nivel) res ← copiar(listDeTipo(s, "comercio"))
3: **for**(nat i ← 0; i < long(s.uniones); i ← i + 1) :
4: itDicc(Pos, Nivel) itCs ← crearIt(**comerciosAux**(*s.uniones[i].sc), casasTotales)
5: **while**(haySiguiente(itCs)) :
6: Pos p ← siguienteClave(itCs)
7: Nivel n ← siguienteSignificado(itCs)
8: **if**(¬def?(res, p) ∧ ¬def?(casasTotales, p)) :
9: definirRapido(res, p, s.turno - s.uniones[i].turnoUnido + n)
10: avanzar(itCs)
11: **return** res
Complejidad: O(sumConstrucciones^2 * nodos + nodos)
Justificación: El algoritmo es muy similar al de casas por lo tanto su justificacion es muy similar

manhatizar(**inout** dicc(Pos, Nivel) comercios, **in** dicc(Pos, Nivel) casasTotales)
1: itDicc(Pos, Nivel) itCs ← crearIt(comercios)
2: **while**(haySiguiente(itCs)) :
3: Pos p ← siguienteClave(itCs)
4: Nivel n ← siguienteSignificado(itCs)
5: definir(comercios, p, max(n, nivelCom(p, casasTotales)))
Complejidad: O(sumConstrucciones^2)

---

nivelCom(**in** Pos p, **in** dicc(pos, Nivel) cs) → Nat
1: nat maxLvl ← 1
2: **for**(int i = -3; i ≤ 3; ++i) :
3: **for**(int j = |i|-3; j ≤ 3-|i|; ++j) :
4: **if**(p.x + i ≥ 0 ∧ p.y + j ≥ 0) :
5: Pos p2 ← <p.x+i, p.y+j>
6: **if**(def?(cs, p2)) :
7: maxLvl = max(maxLvl, obtener(cs, p2))
8: **return** maxLvl
Complejidad: O(#claves(cs))

---

popularidad(**in** SimCity s) → res : Nat
1: **return** s.popularidad
Complejidad: O(1)

---

turnos(**in** SimCity s) → res : Nat
1: **return** s.antiguedad
Complejidad: O(1)

• ∪ •(**in** dicc(α, β) d1, **in** dicc(α, β) d2) → res : dicc(α, β)
1: dicc(α, β) res = copiar(d1)
2: itDicc(α, β) itCs ← crearIt(d2);
3: **while**(haySiguiente(itCs)) :
4: α a ← siguienteClave(itCs)
5: β b ← siguienteSignificado(itCs)
6: **if**(¬def?(res, a)) :
7: definirRapido(res, a, b)
8: avanzar(itCs)
9: **return** res
Complejidad: O(copy(d1) + #claves(d2))

2.4. Módulo Servidor

2.4.1. Interfaz

Interfaz

usa: SimCity, Mapa, Pos, Nombre, Construcción, Diccionario Trie, Diccionario Lineal

exporta: todo

se explica con: SERVIDOR

géneros: `server`

Operaciones básicas de server

Sea S: *servidor*, donde N es la cantidad de partidas definidas en S, nom_i es el nombre de la partida i y sc_i es el SimCity asociado a nom_i.

NUEVOSERVER() $\rightarrow$ *res* : `server`
Pre $\equiv$ {true}
Post $\equiv$ {$r\hat{e}s$ $=_{\text{obs}}$ nuevoServer}
Complejidad: **O**(1)
Descripción: Crea un servidor.
Aliasing: No tiene.

PARTIDAS(**in** s: `server`) $\rightarrow$ *res* : `diccTrie(Nombre, SimCity)`
Pre $\equiv$ {true}
Post $\equiv$ {$r\hat{e}s$ $=_{\text{obs}}$ partidas($\hat{s}$)}
Complejidad: **O**($\sum_{i=0}^{N} copy(nom_i) + copy(sc_i)$)
Descripción: Devuelve un diccionario con todas las partidas del servidor.
Aliasing: Por copia.

CONGELADAS(**in** s: `server`) $\rightarrow$ *res* : `conj(Nombre)`
Pre $\equiv$ {true}
Post $\equiv$ {$r\hat{e}s$ $=_{\text{obs}}$ congeladas($\hat{s}$)}
Complejidad: **O**($\sum_{i=0}^{N} copy(nom_i)$)
Descripción: Devuelve el conjunto con los nombres de las partidas no modificables.
Aliasing: Por copia.

NUEVAPARTIDA(**in/out** s: `server`, **in** p: `Nombre`, **in** m: `Mapa`)
Pre $\equiv$ {$\hat{s}$ $=_{\text{obs}}$ s_0 $\wedge$ $\neg$def?($\hat{p}$, partidas($\hat{s}$))}
Post $\equiv$ {$\hat{s}$ $=_{\text{obs}}$ nuevaPartida(s_0, $\hat{p}$, $\hat{m}$)}
Complejidad: **O**(copy(p) + copy(m))
Descripción: Agrega una partida nueva al servidor.
Aliasing: No tiene.

UNIRPARTIDAS(**in/out** s: `server`, **in** $p1$: `Nombre`, **in** $p2$: `Nombre`)
Pre $\equiv$ {unionValida[1]($\hat{s}$, $\hat{p1}$, $\hat{p2}$) $\wedge$ $\hat{s} = s_0$}
Post $\equiv$ {$\hat{s}$ $=_{\text{obs}}$ unirPartidas(s_0, $\hat{p1}$, $\hat{p2}$)}
Complejidad: **O**($|nombreMasLargo|$)
Descripción: Agrega el SimCity $s2$ asociado a $p2$, al SimCity $s1$, asociado a $p1$. $s2$ pasa a ser no modificable.
Aliasing: Se guarda una referencia a $s2$ en $s1$. Cualquier cambio sobre $s2$ modificará la representación de $s1$. Se garantiza que $s1$ no modificará a $s2$.

AVANZARTURNOPARTIDA(**in/out** s: `server`, **in** p: `Nombre`, **in** cs: `dicc(Pos × Construcción)`)
Pre $\equiv$ {avanzarValido[1]($\hat{s}$, $\hat{p}$, $\hat{cs}$) $\wedge$ $\hat{s}$ $=$ s_0}
Post $\equiv$ {$\hat{s}$ $=_{\text{obs}}$ avanzarTurnoPartida(s_0, $\hat{p}$, $\hat{cs}$)}
Complejidad: **O**($|nombreMasLargo|$)
Descripción: Avanza el turno de una partida y agrega las construcciones definidas en el diccionario de construcciones.
Aliasing: Se guarda un puntero al diccionario en el SimCity. Cualquier cambio en este modificará a la partida.

AGREGARCASA(**in/out** s: `server`, **in** p: `Nombre`, **in** pos: `Pos`)
Pre $\equiv \{\hat{s} = s_0 \wedge \text{agregarValido}^1(\hat{s}, \hat{p}, \hat{pos})\}$
Post $\equiv \{\hat{s} =_{\text{obs}} \text{agregarCasa}(s_0, \hat{p}, \hat{pos})\}$
Complejidad: $\mathbf{O}(|nombreMasLargo|)$
Descripción: Agrega una nueva casa.
Aliasing: No tiene.

AGREGARCOMERCIO(**in/out** s: `server`, **in** $p1$: `Nombre`, **in** $p2$: `Nombre`)
Pre $\equiv \{\hat{s} = s_0 \wedge \text{agregarValido}^1(\hat{s}, \hat{p}, \hat{pos})\}$
Post $\equiv \{\hat{s} =_{\text{obs}} \text{agregarComercio}(s_0, \hat{p}, \hat{pos})\}$
Complejidad: $\mathbf{O}(|nombreMasLargo|)$
Descripción: Agrega un nuevo comercio.
Aliasing: No tiene.

VERPOPULARIDAD(**in** s: `server`, **in** p: `Nombre`) $\rightarrow$ res : `Nat`
Pre $\equiv \{\text{def?}(\hat{p}, \text{partidas}(\hat{s}))\}$
Post $\equiv \{r\hat{e}s =_{\text{obs}} \text{verPopularidad}(\hat{s}, \hat{p})\}$
Complejidad: $\mathbf{O}(|nombreMasLargo|)$
Descripción: Devuelve la popularidad de la partida.
Aliasing: Devuelve una referencia no modificable.

VERTURNO(**in** s: `server`, **in** p: `Nombre`) $\rightarrow$ res : `Nat`
Pre $\equiv \{\text{def?}(\hat{p}, \text{partidas}(\hat{s}))\}$
Post $\equiv \{r\hat{e}s =_{\text{obs}} \text{verTurno}(\hat{s}, \hat{p})\}$
Complejidad: $\mathbf{O}(|nombreMasLargo|)$
Descripción: Devuelve la antiguedad de la partida.
Aliasing: Devuelve una referencia no modificable.

VERMAPA(**in** s: `server`, **in** p: `Nombre`) $\rightarrow$ res : `Mapa`
Pre $\equiv \{\text{def?}(\hat{p}, \text{partidas}(\hat{s}))\}$
Post $\equiv \{r\hat{e}s =_{\text{obs}} \text{verMapa}(\hat{s}, \hat{p})\}$
Complejidad: $\mathbf{O}(|nombreMasLargo|) + \mathbf{O}(\text{mapa}(sc))$, donde sc es el SimCity asocidado a p
Descripción: Devuelve el mapa de la partida.
Aliasing: Devuelve una copia.

VERCASAS(**in** s: `server`, **in** p: `Nombre`) $\rightarrow$ res : `dicc(Pos, Nat)`
Pre $\equiv \{\text{def?}(\hat{p}, \text{partidas}(\hat{s}))\}$
Post $\equiv \{r\hat{e}s =_{\text{obs}} \text{verCasas}(\hat{s}, \hat{p})\}$
Complejidad: $\mathbf{O}(|nombreMasLargo|) + \mathbf{O}(\text{casas}(sc))$, donde sc es el SimCity asocidado a p
Descripción: Devuelve un diccionario con las posiciones y niveles de las casas de la partida.
Aliasing: Devuelve una copia.

VERCOMERCIOS(**in** s: `server`, **in** p: `Nombre`) $\rightarrow$ res : `dicc(Pos, Nat)`
Pre $\equiv \{\text{def?}(\hat{p}, \text{partidas}(\hat{s}))\}$
Post $\equiv \{r\hat{e}s =_{\text{obs}} \text{verComercios}(\hat{s}, \hat{p})\}$
Complejidad: $\mathbf{O}(|nombreMasLargo|) + \mathbf{O}(\text{comercios}(sc))$, donde sc es el SimCity asocidado a p
Descripción: Devuelve un diccionario con las posiciones y niveles de los comercios de la partida
Aliasing: Devuelve una copia.

1. definido en Definiciones Auxilares de Servidor.

2.4.2. Representación

Representación

Representación de servidor

Un servidor almacena y actualiza los diferentes SimCity. Se representa como un diccionario implementado en un trie, donde las claves son los nombres de las partidas y los significados un puntero al SimCity y su estado (si es modificable o no).

Elegimos esta estructura para cumplir con las restricciones dadas de complejidad. Buscar una clave en el diccTrie está acotado por la clave más larga definida en el mismo. Con lo cual, todas las operaciones del servidor en relación a una partida específica serán por lo menos $\mathbf{O}(|nombreMasLargo|)$.

`servidor` **se representa con** `diccTrie(Nombre, partida)`

donde `partida` es `tupla(`*modificable* : `bool` ,
sim : `puntero(SimCity)`)

Rep : estr $\longrightarrow$ boolean
Rep(e) $\equiv$ true $\iff$
($\forall$ *partida*$_1$, *partida*$_2$: Nombre)
((def?(*partida*$_1$, e) $\wedge$ def?(*partida*$_2$, e) $\wedge$ *partida*$_1 \neq$ *partida*$_2$) $\Rightarrow_{\text{L}}$
obtener(*partida*$_1$, e).sim $\neq$ obtener(*partida*$_2$, e).sim
) $\wedge$
($\forall$ *partida*: Nombre)(def?(*partida*, e) $\Rightarrow_{\text{L}}$ obtener(*partida*, e).sim $\neq$ NULL)

Abs : estr e $\longrightarrow$ servidor {Rep(e)}
Abs(e) $\equiv$ s: servidor |
($\forall$ *nombre*: Nombre)
(*nombre* $\in$ congelados(s) $\Leftrightarrow$
(def?(*nombre*, e) $\wedge_{\text{L}}$ $\neg$ obtener(*nombre*, e).modificable))
$\wedge$
($\forall$ *nombre*: Nombre)
(def?(*nombre*, partidas(s)) $\Leftrightarrow$ def?(*nombre*, e))
$\wedge_{\text{L}}$
($\forall$ *nombre*: Nombre)
(def?(*nombre*, partidas(s)) $\Rightarrow_{\text{L}}$
obtener(*nombre*, partidas(s)) $=_{\text{obs}}$ *(obtener(*nombre*, e).sim))

2.4.3. Implementación

Algoritmos

---

nuevoServer() → *res* : estr

1: $res \leftarrow vacio()$
2: **return** *res*

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

---

partidas(**in** *e* : `estr`) → *res* : diccTrie(Nombre, SimCity)

1: $res \leftarrow vacio()$
2: $it \leftarrow crearIt(e)$
3: $while(haySiguiente(it))$:
4: $\quad nom \leftarrow siguienteClave(it)$
5: $\quad sc \leftarrow *(siguienteSignificado(it).sim)$
6: $\quad definirRapido(res,\ nom,\ sc)$
7: $\quad avanzar(it)$
8: **return** *res*

Complejidad: $\mathbf{O}(\sum_{i=0}^{N}(copy(nom_i) + copy(sc_i)))$

---

congeladas(**in** *e* : `estr`) → *res* : conj(Nombre)

1: $res \leftarrow vacio()$
2: $it \leftarrow crearIt(e)$
3: $while(haySiguiente(it))$:
4: $\quad if\ (\neg siguienteSignificado(it).modificable)$:
5: $\quad\quad nom \leftarrow siguienteClave(it)$
6: $\quad\quad agregarRapido(res,\ nom)$
7: $\quad avanzar(it)$
8: **return** *res*

Complejidad: $\mathbf{O}(\sum_{i=0}^{N} copy(nom_i))$

---

nuevaPartida(**in/out** *e* : `estr`, **in** *p* : `Nombre`, **in** *m* : `Mapa`)

1: $definirRapido(e,\ p,\ \langle true,\ \&(iniciar(m))\rangle)$ ▷ Reservamos memoria para el nuevo SimCity

Complejidad: $\mathbf{O}(\text{copy}(p) + \text{copy}(m))$

---

unirPartidas(**in/out** *e* : `estr`, **in** *p*1 : `Nombre`, **in** *p*2 : `Nombre`)

1: $definir(e,\ p1, \langle true,\ \&(unir(*(significado(p1,\ e).sim),\ *(significado(p2,\ e).sim)))\rangle)$
2: $definir(e,\ p2, \langle false,\ significado(p2,\ e).sim\rangle)$

Complejidad: $\mathbf{O}(|nombreMasLargo|)$
Justificacion: $unir \in \mathbf{O}(1) + buscar\ en\ diccTrie \in \mathbf{O}(|nombreMasLargo|) \rightarrow \mathbf{O}(|nombreMasLargo|)$

---

avanzarTurnoPartida(**in/out** *e* : `estr`, **in** *p* : `Nombre`, **in** *cs* : `diccLineal(Pos × Construccion))`

1: $definir(e,\ p, \langle true,\ \&(avanzarTurno(*(significado(p,\ e).sim),\ cs))\rangle)$ ▷ cs por referencia

Complejidad: $\mathbf{O}(|nombreMasLargo|)$
Justificacion: $avanzarTurno \in \mathbf{O}(1) + buscar\ en\ diccTrie \in \mathbf{O}(|nombreMasLargo|)$
$\rightarrow \mathbf{O}(|nombreMasLargo|)$

---

agregarCasa(**in**/**out** s : `estr`, **in** *partida* : `String`, **in** *pos* : `Pos`)

1: $cs \leftarrow definirRapido(vacio(),\ pos,\ "casa")$
2: $definir(e,\ p, \langle true,\ \&(avanzarTurno(*(significado(p,\ e).sim),\ cs))\rangle)$

Complejidad: $\mathbf{O}(|nombreMasLargo|)$

agregarComercio(**in**/**out** s : `estr`, **in** *partida* : `String`, **in** *pos* : `Pos`)

1: $cs \leftarrow definirRapido(vacio(),\ pos,\ "comercio")$
2: $definir(e,\ p, \langle true,\ \&(avanzarTurno(*(significado(p,\ e).sim),\ cs))\rangle)$

Complejidad: $\mathbf{O}(|nombreMasLargo|)$

verMapa(**in** s : `estr`, **in** *partida* : `Nombre`) → *res* : Mapa

1: $sc \leftarrow obtener(partida,\ s).sim$
2: $res \leftarrow mapa(*sc)$
3: **return** *res*

Complejidad: $\mathbf{O}(|nombreMasLargo|) + \mathbf{O}(mapa(*sc))$

verCasas(**in** s : `estr`, **in** *partida* : `Nombre`) → *res* : DiccLineal(Pos, Nivel)

1: $sc \leftarrow obtener(partida,\ s).sim$
2: $res \leftarrow casas(*sc)$
3: **return** *res*

Complejidad: $\mathbf{O}(|nombreMasLargo|) + \mathbf{O}(casas(*sc))$

verComercios(**in** s : `estr`, **in** *partida* : `Nombre`) → *res* : DiccLineal(Pos, Nivel)

1: $sc \leftarrow obtener(partida,\ s).sim$
2: $res \leftarrow comercios(*sc)$
3: **return** *res*

Complejidad: $\mathbf{O}(|nombreMasLargo|) + \mathbf{O}(comercios(*sc))$

verPopularidad(**in** s : `estr`, **in** *partida* : `Nombre`) → *res* : Nat

1: $sc \leftarrow obtener(partida,\ s).sim$
2: $res \leftarrow popularidad(*sc)$
3: **return** *res*

Complejidad: $\mathbf{O}(|nombreMasLargo|)$

verTurno(**in** s : `estr`, **in** *partida* : `Nombre`) → *res* : Nat

1: $sc \leftarrow obtener(partida,\ s).sim$
2: $res \leftarrow turnos(*sc)$
3: **return** *res*

Complejidad: $\mathbf{O}(|nombreMasLargo|)$