DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1: Diseño

SimCity

1 de junio de 2022 — Algoritmos y Estructuras de Datos 2

Grupo 27

Integrante	LU	Correo electrónico
Acha, Francisco	433/19	facha@dc.uba.ar
Dacko, Maximiliano	284/21	mdacko@dc.uba.ar
Freire, Guido	978/21	gfreire@dc.uba.ar
Marino, María	450/21	mamarino@dc.uba.ar

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

Índice

1. Especificación

1.1. TAD Mapa

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 generadores, observadores

usa Conj, Nat

observadores básicos

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

generadores

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

otras operaciones

$\bullet \cup \bullet$: Mapa $m1$ $\times$ Mapa $m2$ $\longrightarrow$ Mapa

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

horizontales(crear(hs, vs)) $\equiv$ hs
verticales(crear(hs, vs)) $\equiv$ vs
a $\cup$ b $\equiv$ crear(horizontales(a) $\cup$ horizontales(b), verticales(a) $\cup$ verticales(b))

Fin TAD

1.2. TAD SimCity

TAD Pos **es** Tupla(Int, Int)
TAD Construccion **es** String
TAD Nivel **es** Nat

TAD SIMCITY

igualdad observacional

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

géneros SimCity

exporta generadores, observadores, proposiciones auxiliares

usa Dicc, Nat, Conj, Mapa, Pos, Nivel, 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	{preAvanzarTurno(s, cs)}
unir	:	SimCity a $\times$ SimCity b	$\longrightarrow$	SimCity	{preUnir(a,b)}

otras operaciones

turnos	:	SimCity	$\longrightarrow$	Nat	
construcciones	:	SimCity	$\longrightarrow$	dicc(Pos, Nivel)	
construccionesAux	:	dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Nivel)	$\longrightarrow$	dicc(Pos, Nivel)	
nivelComercio	:	SimCity $\times$ Pos	$\longrightarrow$	Nat	
maxNivel	:	conj(Pos) ps $\times$ dicc(Pos $\times$ Nivel) cs	$\longrightarrow$	Nivel	{$\neg$ $\emptyset$?(ps)}
unirCasasAux	:	conj(Pos) ca $\times$ conj(Pos) cb $\times$ dicc(Pos $\times$ Nivel) a $\times$ dicc(Pos $\times$ Nivel)	$\longrightarrow$	dicc(Pos, Nivel)	$\left\{\begin{array}{l}(\forall\ \text{p} : \text{Pos})(\text{p} \in \text{ca} \Rightarrow \text{def?(p, a)}) \wedge \\ (\forall\ \text{p} : \text{Pos})(\text{p} \in \text{cb} \Rightarrow \text{def?(p, b)})\end{array}\right\}$
unirComerciosAux	:	conj(Pos) ca $\times$ conj(Pos) cb $\times$ dicc(Pos $\times$ Nivel) a $\times$ dicc(Pos $\times$ Nivel)	$\longrightarrow$	dicc(Pos, Nivel)	$\left\{\begin{array}{l}(\forall\ \text{p} : \text{Pos})(\text{p} \in \text{ca} \Rightarrow \text{def?(p, a)}) \wedge \\ (\forall\ \text{p} : \text{Pos})(\text{p} \in \text{cb} \Rightarrow \text{def?(p, b)})\end{array}\right\}$
posMaxConstruccion	:	SimCity s	$\longrightarrow$	conj(Pos)	{turno(s) > 0}
damePosCon	:	dicc(Pos $\times$ Nivel) cs	$\longrightarrow$	Pos	{$\neg\emptyset$?(cs)}
casillasCercanas	:	Pos	$\longrightarrow$	conj(Pos)	

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

mapa(iniciar(m)) $\equiv$ m

mapa(avanzarTurno(s, cs)) $\equiv$ mapa(s)

mapa(unir(a, b)) $\equiv$ mapa(a) $\cup$ mapa(b)

casas(iniciar(m)) $\equiv$ vacío()

casas(avanzarTurno(s, cs)) $\equiv$
if #claves(cs) = 1 **then**
  if obtener(dameUno(claves(cs)), cs) = 'casa' **then**
    definir(dameUno(claves(cs)), 1, valoresMasUno(casas(s)))
  else
    valoresMasUno(casas(s))
  fi
else
  if obtener(dameUno(claves(cs)), cs) = 'casa' **then**
    definir(dameUno(claves(cs)), 0, casas(avanzarTurno(s,
      borrar(dameUno(claves(cs)), cs))))
  else
    casas(avanzarTurno(s, borrar(dameUno(claves(cs)), cs)))
  fi
fi

casas(unir(a, b)) $\equiv$ unirCasasAux(casas(a), casas(b), a, b)

comercios(iniciar(m)) $\equiv$ vacío()

comercios(avanzarTurno(s, cs)) ≡ **if** #claves(cs) = 1 **then**
if obtener(dameUno(claves(cs)), cs) = 'comercio' **then**
definir(dameUno(claves(cs)), nivelComercio(s, dameUno(claves(cs)), valoresMasUno(comercios(s))))
else
valoresMasUno(comercios(s))
fi
else
if obtener(dameUno(claves(cs)), cs) = 'comercio' **then**
definir(dameUno(claves(cs)), nivelComercio(s, dameUno(claves(cs)), comercios(avanzarTurno(s, borrar(dameUno(claves(cs)), cs))))
else
comercios(avanzarTurno(s, borrar(dameUno(claves(cs)), cs)))
fi
fi

comercios(unir(a, b)) ≡ unirComerciosAux(comercios(a), comercios(b), a, b)

popularidad(iniciar(m)) ≡ 0

popularidad(avanzarTurno(s, cs)) ≡ popularidad(s)

popularidad(unir(a, b)) ≡ popularidad(a) + 1

turnos(iniciar) ≡ 0

turnos(avanzarTurno(s, cs)) ≡ turnos(s) + 1

turnos(unir(s, s')) ≡ max(turnos(s), turnos(s'))

construcciones(s) ≡ construccionesAux(casas(s), comercios(s))

construccionesAux(cs1, cs2) ≡ **if** vacío?(cs1) **then**
cs2
else
definir(
dameUno(claves(cs1)),
obtener(dameUno(claves(cs1)), cs1),
construccionesAux(sinUno(cs1), cs2))
fi

nivelComercio(s, p) ≡ maxNivel(casillasCercanas(p) ∩ claves(casas(s)), casas(s))

maxNivel(ps, cs) ≡ **if** #ps = 0 **then**
1
else
max(obtener(dameUno(ps), cs), maxNivel(sinUno(ps), cs))
fi

unirCasasAux(ca, cb, a, b) ≡ **if** ∅?(cb) **then**
ca
else
if def?(damePosCon(cb), casas(a)) **then**
definir(damePosCon(cb),
max(obtener(damePosCon(cb), casas(a)), obtener(damePosCon(cb), casas(b)), unirCasasAux(ca, sinCasa(cb), a, b))
else
if def?(damePosCon(cb), comercios(b)) **then**
unirCasasAux(ca, sinCasa(cb), a, b)
else
definir(damePosCon(cb), obtener(damePosCon(cb), casas(b)), unirCasasAux(ca, sinCasa(cb), a, b))
fi
fi
fi

```
unirComerciosAux(ca, cb, a, b) ≡ if ∅?(cb) then
                                   ca
                                 else
                                   if def?(damePosCon(cb), casas(a)) then
                                     definir(damePosCon(cb),
                                     maxNivel(casilasCercanas(damePosCon(cb)), casas(unir(a, b))),
                                     unirComerciosAux(ca, sinComercio(cb), a, b))
                                   else
                                     if def?(damePosCon(cb), comercios(b)) then
                                       definir(damePosCon(b),
                                       max(obtener(damePosCon(b), comercios(b)), obtener(damePosCon(b), comercios(a))),
                                       unirComerciosAux(ca, sinComercio(cb), a, b))
                                     else
                                       definir(damePosCon(cb),
                                       obtener(damePosCon(cb), comercios(b)),
                                       unirComerciosAux(ca, sinComercio(cb), a, b))
                                     fi
                                   fi
                                 fi
posMaxConstruccion(s)          ≡ posMaxConstruccionAux(construcciones(s), turno(s))
posMaxConstruccionAux(cc, max) ≡ if vacío?(cc) then vacío()
                                 else if obtener(dameUno(claves(cc)), cc) = max then
                                   Ag(dameUno(claves(cc)), posMaxConstruccionAux(sinUno(cc), max))
                                 else
                                   posMaxConstruccionAux(sinUno(cc), max)
                                   fi
                                 fi
damePosCon(cs)                 ≡ dameUno(claves(cs))
casillasCercanas(p, sc)        ≡ if ∅?(claves(construcciones(sc)))
                                 then
                                   ∅

                                 else
                                   if distancia(p, dameUno(claves(construcciones(sc)))) = 3
                                   then
                                     Ag(dameUno(claves(construcciones(sc))),
                                     casillasCercanas(p, sinUno(claves(construcciones(sc)))))
                                   else
                                     casillasCercanas(p, sinUno(claves(construcciones(sc))))
                                   fi
                                 fi
distancia(p_0, p_1)            ≡ |π_0(p_2) − π_0(p_1)| + |π_1(p_2) − π_1(p_1)|
valoresMasUno(d)               ≡ if vacío?(d) then
                                   vacío()
                                 else
                                   definir(valoresMasUno(borrar(d, dameUno(claves(d)))),
                                   dameUno(claves(d)),
                                   obtener(d, dameUno(claves(d))) + 1)
                                 fi
```

proposiciones auxiliares

preUnir(a, b) $\equiv$ ($\forall$ p : Pos) (p $\in$ (claves(construcciones(a)) $\cup$ claves(construcciones(b))) $\Rightarrow$
$\pi_0(p) \notin$ (verticales(mapa(a)) $\cup$ verticales(mapa(b))) $\wedge$
$\pi_1(p) \notin$ (horizontales(mapa(a)) $\cup$ horizontales(mapa(b))))
posMaxConstruccion(a) $\cap$ claves(construcciones(b)) $= \emptyset \wedge$
posMaxConstruccion(b) $\cap$ claves(construcciones(a)) $= \emptyset$

preAvanzarTurno(s, cs) $\equiv$ ($\forall$ p : Pos)(p $\in$ claves(cs) $\Rightarrow$
$\pi_1(p) \notin$ horizontales(mapa(s)) $\wedge$
$\pi_0(p) \notin$ verticales(mapa(s)) $\wedge$ p $\notin$ (claves(construcciones(s)))) $\wedge \neg\emptyset?$(cs))

Fin TAD

1.3. TAD Servidor

TAD Construccion **es** String
TAD Nombre **es** String

TAD SERVIDOR

igualdad observacional

$$(\forall s, s' : \text{Servidor}) \left(s =_{\text{obs}} s' \iff \begin{pmatrix} \text{SimCitys(s)} =_{\text{obs}} \text{SimCitys(s')} \wedge \\ \text{Uniones(s)} =_{\text{obs}} \text{Uniones(s')} \end{pmatrix} \right)$$

géneros Servidor

exporta generadores, observadores

usa Conj, SimCity, Dicc, Tupla, Construccion, Nombre

observadores básicos

SimCitys : Servidor $\longrightarrow$ dicc(Nombre, SimCity)
Uniones : Servidor $\longrightarrow$ conj(tupla(Nombre, Nombre))

generadores

crearServidor : $\longrightarrow$ Servidor
agregarSimCity : Servidor $\times$ Nombre sc $\longrightarrow$ Servidor
{sc $\notin$ claves(SimCitys(s))}
avanzarTurnoS : Servidor s $\times$ Nombre sc $\times$ Dicc(Pos $\times$ Construccion) cs $\longrightarrow$ Servidor
$\left\{ \begin{matrix} \text{sc} \in \text{claves(SimCitys(s))} \wedge \text{sc} \notin \text{YaUnidos(s, Uniones(s))} \wedge \\ \text{PreAvanzarTurno(obtener(sc, claves(SimCitys(s))))} \end{matrix} \right\}$
unirS : Servidor s $\times$ Nombre $sc1$ $\times$ Nombre $sc2$ $\longrightarrow$ Servidor
$\left\{ \begin{matrix} \text{sc1} \neq \text{sc2} \wedge \text{sc1, sc2} \in \text{claves(SimCitys(s))} \wedge \text{sc1} \notin \text{YaUnidos(s, Uniones(s))} \wedge \\ \text{PreUnir(obtener(sc1, SimCitys(s)), obtener(sc2, SimCitys(s)))} \end{matrix} \right\}$

otras operaciones

YaUnidos : Servidor $\times$ conj(tupla(Nombre $\times$ Nombre)) $\longrightarrow$ conj(Nombre)

axiomas $\forall s$: Servidor, $\forall sc, sc1, sc2$: Nombre, $\forall cs$: dicc(Pos, Construccion)

SimCitys(crearServidor()) $\equiv \emptyset$
SimCitys(agregarSimCity(s, sc)) $\equiv$ definir(sc, iniciar(), SimCitys(s))
SimCitys(avanzarTurnoS(s, sc, cs))) $\equiv$ definir(sc, avanzarTurno(obtener(sc, SimCitys(s)), cs), SimCitys(s))
SimCitys(unirS(s, sc1, sc2)) $\equiv$ definir(sc1, unir(obtener(sc1, SimCitys(s)), obtener(sc2, SimCitys(s))), SimCitys(s))
Uniones(crearServidor()) $\equiv \emptyset$
Uniones(agregarSimCity(s, sc, cs) $\equiv$ Uniones(s)
Uniones(avanzarTurnoS(s, sc, cs)) $\equiv$ Uniones(s)
Uniones(unirS(s, sc1, sc2)) $\equiv$ Ag(<sc1, sc2>, Uniones(s))

YaUnidos(s, us) $\equiv$ **if** $\#(\text{us}) == 0$ **then**
$\emptyset$
else
$\text{Ag}(\pi_1(\text{dameUno(us)}), \text{YaUnidos(s, SinUno(us))})$
fi

Fin TAD

2. Módulos de referencia

2.1. Módulo Mapa

Interfaz

se explica con: MAPA

géneros: `mapa`

Operaciones básicas de mapa

CREARMAPA(**in** hs: `conj(Nat)`,**in** vs: `conj(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

HAYRIO?(**in** m: `Mapa`,**in** p: `Pos`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \pi_0(p) \in verticales(m) \vee \pi_1(p) \in horizontales(m)\}$
Complejidad: $O(\#verticales(m) + \#horizontales(m))$
Descripción: dada una posición indica si allí hay un río

UNIRMAPA(**in/out** $m1$: `Mapa`, **in** $m2$: `Mapa`)
Pre $\equiv$ $\{m1 = M1_0\}$
Post $\equiv$ $\{m1 = m1 \cup m2\}$
Complejidad: $O(\#m1.verticales \cdot \#m2.verticales + \#m1.horizontales \cdot \#m2.horizontales)$
Descripción: une dos mapas, copiando los ríos de m2 a m1

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

---

iCrearMapa(**in** hs: `conj(Nat)`, **in** vs: `conj(Nat)`) $\rightarrow$ res : estr

1: res $\leftarrow$ <horizontales: hs, verticales: vs>

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

---

iHayRío?(**in** p: `Pos`, **in** m: `estr`) $\rightarrow$ res : bool

1: res $\leftarrow$ Pertenece?(m.verticales, $\pi_0(p)$) $\vee$ Pertenece?(m.horizontales, $\pi_1(p)$)

Complejidad: $O(\#m.verticales + \#m.horizontales)$

---

iUnirMapa(**in/out** $m1$: `estr`, **in** $m2$: `estr`)

```
1: it = CrearIt(m2.horizontales)
2: while HaySiguiente(it) do
3:     Agregar(m1.horizontales, Siguiente(it))
4:     Avanzar(it)
5: end while
6: it = CrearIterador(m2.verticales)
7: while HaySiguiente(it) do
8:     Agregar(m1.verticales, Siguiente(it))
9:     Avanzar(it)
10: end while
```

Complejidad: $O(\#m1.verticales \cdot \#m2.verticales + \#m1.horizontales \cdot \#m2.horizontales)$

2.2. Módulo SimCity

Interfaz

se explica con: SimCity

géneros: `SimCity`

Operaciones básicas de SimCity

INICIAR(**in** m : `mapa`) $\rightarrow$ res : `SimCity`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} iniciar(m)\}$
Complejidad: $O(copy(m))$
Descripción: crea un SimCity con el mapa pasado como argumento
Aliasing: no genera aliasing

AVANZARTURNO(**in/out** s : `SimCity`, **in** *casas* : `conj(Pos)`, **in** *comercios* : `conj(Pos)`)
Pre $\equiv$ $\{(s =_{\text{obs}} S_0 \wedge$
$\neg$ vacío?(casas $\cup$ comercios)) $\wedge_L$
$(\forall$ p : Pos)(p $\in$ claves(casas $\cup$ comercios) $\Rightarrow_L$
$(\pi_0$(p) $\notin$ verticales(mapa(s)) $\wedge$
π_1(p) $\notin$ horizontales(mapa(s))) $\wedge$
p $\notin$ claves(construcciones(s)))$\}$
Post $\equiv$ $\{s =_{\text{obs}} avanzarTurno(S_0, cs)\}$
Complejidad: $O(\#(casas) + \#(comercios))$
Descripción: avanza un turno en el SimCity pasado como argumento, agregando las construcciones indicadas por el usuario en el diccionario *cs*
Aliasing: no genera aliasing

MAPA(**in** s : `SimCity`) $\rightarrow$ res : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} mapa(s)\}$
Complejidad: $O(\#e.uniones \cdot r)$
donde r es la máxima cantidad de ríos de los mapas de los SimCitys que forman parte de *e.uniones*
Descripción: devuelve una copia del mapa asociado al SimCity
Aliasing: no genera aliasing ya que el resultado se devuelve por copia

CASAS(**in** s : `SimCity`) $\rightarrow$ res : `dicc(Pos, Turno)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} casas(s)\}$
Complejidad: $O(copy(Pos) \cdot \#e.uniones \cdot m))$
donde m es la máxima cantidad de casas de los SimCitys que forman parte de *e.uniones*.
Descripción: devuelve las casas de s, tanto las del juego original como las de los juegos unidos a éste
Aliasing: las casas se devuelven por copia

COMERCIOS(**in** s: `SimCity`) → res : `dicc(Pos, Turno)`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} comercios(s)$}
Complejidad: $O(copy(Pos) \cdot \#e.uniones \cdot n)$
Donde n es la maxima cantidad de comercios que tiene un simcity de uniones
Descripción: devuelve los comercios de s, tanto los del juego original como los de los juegos unidos a éste
Aliasing: los comercios se devuelven por copia

NIVELCOMERCIO(**in** s: `SimCity`, **in** p: `Pos`) → res : `Nat`
Pre ≡ {$p \in claves(comercios(s))$}
Post ≡ {$res =_{\text{obs}} nivelComercio(s, p)$}
Complejidad: $O(\#claves(casas(s)) + \#claves(comercios(s))$
Descripción: devuelve el nivel de un comercio en una posición
Aliasing: no genera aliasing

NIVELCASA(**in** s: `SimCity`, **in** p: `Pos`) → res : `Nat`
Pre ≡ {$p \in claves(casas(s))$}
Post ≡ {$res =_{\text{obs}} nivelCasa(s, p)$}
Complejidad: $O(\#claves(e.casas))$
Descripción: devuelve el nivel de una casa en una posición
Aliasing: no genera aliasing

UNIR(**in** a: `SimCity`, **in** b: `SimCity`) → res : `SimCity`
Pre ≡ {(∀ p : Posicion)(p ∈ $claves(casas(a)) \vee p \in claves(comercios(a)) \Rightarrow_L \neg hayRio?(mapa(b, p))) \wedge_L$
(∀ p : Posicion)(p ∈ $claves(casas(b)) \vee p \in claves(comercios(b)) \Rightarrow_L \neg hayRio?(mapa(a, p)))$}
Post≡ {$res =_{\text{obs}} unir(a, b)$}
Complejidad: $O(1)$
Descripción: une el SimCity b al SimCity a
Aliasing: genera aliasing, ya que crea un puntero al SimCity b

Representación

Representación de SimCity

Un SimCity posee un turno actual, contiene construcciones (tanto casas como comercios), cada una de ellas construída en un turno específico; un mapa con sus respectivos ríos y una lista de los SimCities con los que fue unido (de haberlos).

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

donde `estr` es `tupla`(*uniones*: `lista(puntero(SimCity))` , *turnoActual*: `Turno` , *casas*: `diccLineal(Pos, Turno)` , *comercios*: `diccLineal(Pos, Turno)` , *mapa*: `Mapa`)

`Pos` **se representa con** `p`

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

`Turno` **se representa con** `Nat`

Rep : estr e ⟶ bool
Rep(e) ≡ true ⟺ (∀ p : Pos)(p ∈ casas(e) ⇒$_L$ nivelCasa() ≤ e.turnoActual ∧
π_0(p) ∉ verticales(mapa(e)) ∧ π_1(p) ∉ horizontales(mapa(e)) ∧
(∀ p : Pos)(p ∈ claves(comercios(e)) ⇒$_L$ obtener(p, comercios(e)) ≤ e.turnoActual) ∧
π_0(p) ∉ verticales(mapa(e)) ∧ π_1(p) ∉ horizontales(mapa(e)) ∧
claves(casas(e)) ∩ claves(comercios(e)) = ∅ ∧
(∀ s : puntero(SimCity))(s ∈ e.uniones ⇒$_L$ NoEsCircular(&e, *s))

predicados auxiliares

NoEsCircular(S_0, S) $\equiv$ ($\forall$ sc : puntero(SimCity))
($sc \in S.Uniones \Rightarrow_L S_0 \notin *(sc).Uniones \wedge$ NoEsCircular(S_0, sc))

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

Abs(e) $\equiv$ s: SimCity / mapa(s) $=_{obs}$ mapa(e) $\wedge$
claves(casas(s)) $=_{obs}$ claves(casas(e)) $\wedge$ claves(comercios(s)) $=_{obs}$ claves(comercios(e)) $\wedge_L$
($\forall$ p : Pos)(p $\in$ claves(casas(s)) $\Rightarrow_L$ obtener(p, casas(s)) $=_{obs}$ e.turnoActual - obtener(p, casas(e)))
$\wedge$
($\forall$ p : Pos)(p $\in$ claves(comercios(s)) $\Rightarrow_L$ obtener(p, comercios(s)) $=_{obs}$ NivelComercio(e, p)) $\wedge$
popularidad(s) $=_{obs}$ long(e.uniones)

Algoritmos

---

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

```
1: res ← tupla(vacia, 0, vacio, vacio, m)
```

Complejidad: $O(copy(m))$

---

iAvanzarTurno(**in/out** e : `estr`, **in** *casas* : `conjLineal(Pos)`, **in** *comercios* : `conjLineal(Pos)`)

```
1: it ← CrearIt(casas)
2: while HaySiguiente(it) do
3:     DefinirRapido(e.casas, Siguiente(it), e.turnoActual)
4:     Avanzar(it)
5: end while
6: it ← CrearIt(comercios)
7: while HaySiguiente(it) do
8:     DefinirRapido(e.comercios, Siguiente(it), e.turnoActual)
9:     Avanzar(it)
10: end while
```

Complejidad: $O(\#(casas) + \#(comercios))$

---

iMapa(**in** e : `estr`) $\rightarrow$ res : Mapa

```
1: res = e.mapa
2: it ← CrearIt(e.uniones)
3: while HaySiguiente?(it) do
4:     UnirMapa(res, *(Siguiente(it)).mapa)
5:     Avanzar(it)
6: end while
7: return res
```

Complejidad: $O(\#e.uniones * r)$
donde r es la máxima cantidad de ríos de los mapas de los SimCitys que forman parte de $e.uniones$.

---

iCasas(**in** e : `estr`) $\rightarrow$ res : DiccLineal(Pos, Turno)

```
1: res ← e.casas
2: itSimCitys ← CrearIt(e.uniones)
3: while HaySiguiente?(itSimCitys) do
4:     itCasas ← CrearIt(*(Siguiente(itSimCitys)).casas)
5:     while HaySiguiente?(itCasas) do
6:         if Definido?(res, SiguienteClave(itCasas)) then
7:             if SiguienteSignificado(itCasas) <Significado(res, SiguienteClave(itCasas)) then
8:                 Borrar(res, SiguienteClave(itCasas))
9:                 DefinirRapido(res, SiguienteClave(itCasas), SiguienteSignificado(itCasas))
10:            end if
11:        else if ¬ Definido?(Comercios(e), SiguienteClave(itCasas)) then
12:            DefinirRapido(res, SiguienteClave(itCasas), SiguienteSignificado(itCasas))
13:        end if
14:        Avanzar(itCasas)
15:    end while
16:    Avanzar(itSimCitys)
17: end while
18: return res
```

Complejidad: $O((copy(Pos)) * \#(e.uniones) * m)$
donde m es la máxima cantidad de casas de los SimCitys que forman parte de $e.uniones$.

iComercios(**in** e : `estr`) $\rightarrow$ res : DiccLin(Pos, Turno)

```
1: res ← e.comercios
2: itSimCitys ← CrearIt(e.uniones)
3: while HaySiguiente?(itSimCitys) do
4:     itComercios ← CrearIt(*(Siguiente(itSimCitys)).comercios)
5:     while HaySiguiente?(itComercios) do
6:         if Definido?(res, SiguienteClave(itComercios)) then
7:             if SiguienteSignificado(itComercios) < Significado(res, SiguienteClave(itComercios)) then
8:                 Borrar(res, SiguienteClave(itComercios))
9:                 DefinirRapido(res, SiguienteClave(itComercios), SiguienteSignificado(itComercios))
10:            end if
11:        else
12:            DefinirRapido(res, SiguienteClave(itComercios), SiguienteSignificado(itComercios))
13:        end if
14:        Avanzar(itComercios)
15:    end while
16:    Avanzar(itSimCitys)
17: end while
18: return res
```

Complejidad: $O(copy(Pos)) * \#e.uniones * n)$
donde n es la máxima cantidad de comercios de los SimCitys que forman parte de $e.uniones$.

iNivelComercio(**in** s : `estr`,**in** p : `Pos`) $\rightarrow$ res : Nat

```
1:  maxNivel ← 1
2:  it ← CrearIt(casas(s))
3:  while HaySiguiente(it) do
4:      if Distancia(SiguienteClave(it), p) == 3 then
5:          nivel ← e.turnoActual - SiguienteSignificado(it)
6:          if nivel >max then
7:              maxNivel ← nivel
8:          end if
9:      end if
10:     it ← Siguiente(it)
11: end while
12: res ← max(maxNivel, s.turnoActual - Significado(p, comercios(s)))
```

Complejidad: $O(\#claves(casas(s)) + \#claves(comercios(s)))$

iDistancia(**in** p_1, p_2 : `Pos`)$\rightarrow$ res : Nat

```
1: res ← |p1.x - p2.x| + |p1.y - p2.y|
```

Complejidad: $O(1)$

iNivelCasa(**in** s : `estr`, **in** p : `Pos`) $\rightarrow$ res : Nat

```
1: res ← s.turnoActual - Significado(Casas(e), p)
```

Complejidad: $O(O(casas(s)) + \#claves(casas(s)))$

iUnir(**in/out** a : `estr`, **in** b : `estr`)

```
1: AgregarAtras(a.uniones, &b)
2: a.turno ← max(a.turnoActual, b.turnoActual)
```

Complejidad: $O(1)$

2.3. Módulo Servidor

Interfaz

se explica con: SERVIDOR

géneros: `Servidor`

Operaciones básicas de Servidor

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

REGISTRAR(**in/out** *sv* : `Servidor`, **in** *n* : `Nombre`)
Pre $\equiv$ {$\neg Definido?(sv, n)$}
Post$\equiv$ {$res =_{\text{obs}} agregarSimCity(sv, n)$}
Complejidad: $O(|Nombre|)$
Descripción: agrega un SimCity al servidor

AVANZARTURNOS(**in/out** *sv* : `Servidor`, **in** *n* : `Nombre`, **in** *casas* : `conjLineal(Pos)`, **in** *comercios* : `conjLineal(Pos)`)
Pre $\equiv$ {$Definido?(sv, n)$}
Post $\equiv$ {$res =_{\text{obs}} avanzarTurnoS(sv, n)$}
Complejidad: $O(|Nombre| + \#(casas) + \#(comercios))$
Descripción: avanza el turno de un SimCity

UNIRS(**in/out** *sv* : `Servidor`, **in** n_0 : `Nombre`, **in** n_1 : `Nombre`)
Pre $\equiv$ {$Definido?(sv, n_0) \wedge Definido?(sv, n_1)$}
Post $\equiv$ {$res =_{\text{obs}} unirS(sv, n_0, n_1)$}
Complejidad: $O(|Nombre|)$
Descripción: une dos SimCity

Representación

Representación de Servidor

Un Servidor es un diccionario representado en un trie, donde la clave es el nombre del Simcity y el valor es el Simcity per se. Asumimos como dadas las funciones Vacío, Definir, Claves y Significado del módulo diccTrie.

`Servidor` **se representa con** `estr`

donde estr es diccTrie(String, SimCity)

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

Abs : estr e $\longrightarrow$ Servidor {Rep(e)}
Abs(e) $\equiv$ sv : servidor / ($\forall$ n : String)(n $\in$ claves(e) $=_{\text{obs}}$ (def?(n, SimCitys(sv)) $\wedge_L$ obtener(SimCitys(sv), n) $=_{\text{obs}}$ Abs$_{SimCity}$(Significado(e, n))))

Algoritmos

iCrearServidor() $\rightarrow$ *res* : estr

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

Complejidad: $O(1)$

iRegistrar(**in/out** *sv* : `Servidor`, **in** *n* : `Nombre`)

1: DefinirRapido(sv, n, iniciar())

Complejidad: $O(|Nombre|)$

iAvanzarTurnoS(**in/out** *sv* : `Sevidor`, **in** *n* : `Nombre`, **in** *casas* : `conjLineal(Pos)`, **in** *comercios* : `conjLineal(Pos)`)

1: avanzarTurno(Significado(sv, n), casas, comercios)

Complejidad: $O(|Nombre| + \#(casas) + \#(comercios))$

iUnirS(**in/out** *sv* : `Sevidor`, **in** n_0 : `Nombre`, **in** n_1 : `Nombre`)

1: Definir(sv, n_0, Unir(Significado(sv, n_0), Significado(sv, n_1)))

Complejidad: $O(|Nombre|)$

3. Decisiones tomadas

3.1. Conflictos en las uniones

- En el caso de unión que se crucen una casa con otra casa, o un comercio con otro comercio, prevalece la construcción con mayor antigüedad.
- En el caso que se crucen una casa y un comercio, se le da prioridad al comercio.

3.2. Decisiones de diseño

- El nivel inicial de toda construcción es 1.
- La complejidad de agregar una casa o un comercio es $O(|Nombre|)$ como exige el enunciado, en `avanzarTurno` se pueden agregar de a varios, por eso aparece $\#casas + \#comercios$