Algoritmos y Estructuras de Datos II

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo práctico 1 Grupo Algoritmos

Integrante	LU	Correo electrónico
Franco Alesso	464/21	alessofranco@hotmail.com
Leandro Ramirez	90/09	leandroroman.cito@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

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, hayRioEnPos

usa Conj(Nat)

observadores básicos

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

Mapa

generadores

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

otras operaciones

hayRioEnPos : mapa $\times$ Conj(pos) $\longrightarrow$ bool

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

horizontales(crear(hs, vs)) $\equiv$ hs
verticales(crear(hs, vs)) $\equiv$ vs
hayRioEnPos(m,cp) $\equiv$ **if** $\emptyset$?(cp) **then**
false
else
if $\Pi_1(dameUno(cp)) \in horizontales(m) \vee \Pi_2(dameUno(cp)) \in verticales(m)$ **then**
true
else
hayRioEnPos(m,sinUno(cp))
fi
fi

Fin TAD

TAD Pos *es* Tupla(Nat,Nat) **TAD** Nivel *es* Nat

1.2. TAD SimCity

TAD SimCity

igualdad observacional

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

géneros SimCity

exporta generadores, observadores, construcciones, corregComercios, posOcupada, turno, agregoAlgo

usa mapa

observadores básicos

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

generadores

iniciar	: Mapa	$\longrightarrow$	SimCity	
agregarCasaSC	: SimCity s $\times$ Pos p	$\longrightarrow$	Simcity	{¬ posOcupada(s,p)}
agregarComSC	: SimCity s $\times$ Pos p	$\longrightarrow$	Simcity	{¬ posOcupada(s,p))}
avanzarTurno	: SimCity s	$\longrightarrow$	SimCity	{agregoAlgo(s)}
unir	: SimCity a $\times$ SimCity b	$\longrightarrow$	SimCity	

$$\left\{ \begin{array}{l} \neg\text{hayRioEnPos(mapa(b), construcciones(a))} \wedge \neg\text{hayRioEnPos(mapa(a), construcciones(b))} \\ \wedge\ (\forall \text{c:Construccion})(\text{c} \in (\text{construcciones(a)} \cap \text{construcciones(b)}) \rightarrow_L \\ \text{nivel(c,a)} < \text{turno(a)} \wedge \text{nivel(c,b)} < \text{turno(b)} \end{array} \right\}$$

otras operaciones

turno	: SimCity	$\longrightarrow$	Nat
nivel	: Pos p $\times$ Simcity s	$\longrightarrow$	Nat {p $\in$ construcciones(s)}
nivelMaximo	: Nat $\times$ Conj(Pos) c $\times$ dicc(Pos $\times$ Nivel) d	$\longrightarrow$	Nat {c$\subseteq$claves(d)}
unirDicc	: dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Nivel)	$\longrightarrow$	dicc(Pos,Nivel)
incrementar	: conj(Pos) p $\times$ dicc(Pos $\times$ Nivel) d	$\longrightarrow$	dicc(Pos,Nivel) {c$\subseteq$claves(d)}
aDistancia3	: Pos	$\longrightarrow$	Conj(Pos)
corregComercios	: Conj(Pos) c $\times$ dicc(Pos $\times$ Nivel) com $\times$ dicc(Pos $\times$ Nivel) cas	$\longrightarrow$	dicc(Pos.Nivel) {c$\subseteq$claves(com)}
construcciones	: SimCity	$\longrightarrow$	conj(Pos)
posOcupada	: SimCity $\times$ Pos	$\longrightarrow$	bool

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

mapa(iniciar(m))	$\equiv$	m
mapa(agregarCasaSC(s,c))	$\equiv$	mapa(s)
mapa(agregarComSC(s,c))	$\equiv$	mapa(s)
mapa(avanzarTurno(s,d))	$\equiv$	mapa(s)
mapa(unir(a,b))	$\equiv$	crear(horizontales(mapa(a)) $\cup$ horizontales(mapa(b)), verticales(mapa(a)) $\cup$ verticales(mapa(b)))
casas(iniciar(m))	$\equiv$	vacio
casas(agregarCasaSC(s,p))	$\equiv$	definir(p, 0, casas(s))

casas(agregarComSC(s,d)) ≡ casas(s)
casas(avanzarTurno(s)) ≡ incrementar(claves(casas(s)) ,casas(s))
casas(unir(a,b)) ≡ unirDicc(casas(a),casas(b))
comercios(iniciar(m)) ≡ vacio
comercios(agregarCasaSC(s,p)) ≡ comercios(s)
comercios(agregarComSC(s,p)) ≡ corregComercios(claves(comercios(s)) ,definir(p,0,comercios(s), casas(s))
comercios(avanzarTurno(s)) ≡ incrementar(claves(comercios) ,comercios(s))
comercios(unir(a,b)) ≡ corregComercios(claves(comercios(unirDicc(comercios(a),comercios(b)))) , comercios(unirDicc(comercios(a),comercios(b))), casas(s))
popularidad(iniciar(m)) ≡ 0
popularidad(agregarCasaSC(s,p)) ≡ popularidad(s)
popularidad(agregarComSC(s,p)) ≡ popularidad(s)
popularidad(avanzarTurno(s))) ≡ popularidad(s)
popularidad(unir(a,b)) ≡ popularidad(a)+popularidad(b)+1
agregoAlgo(iniciar(m)) ≡ false
agregoAlgo(agregarCasaSC(s,p)) ≡ true
agregoAlgo(agregarComSC(s,p) ≡ true
agregoAlgo(avanzarTurno(s) ≡ false
agregoAlgo(unir(a,b) ≡ agregoAlgo(a)∨ agregoAlgo(b)
turno(s) ≡ nivelMaximo(0, claves(unirDicc(casas(s) , comercios(s))) ,unirDicc(casas(s) , comercios(s)))
nivel(c,s) ≡ **if** def?(c,casas(s)) **then**
obtener(c,casas(s))
else
obtener(c,comercios(s))
fi
nivelMaximo(n,c,d) ≡ **if** ∅?(c) **then**
n
else
if obtener(dameUno(c),d)>n **then**
nivelMaximo(obtener(dameUno(c),d), sinUno(c) ,d)
else
nivelMaximo(n, sinUno(c) ,d)
fi
fi
unirDicc(d_0,d_1) ≡ **if** d_0 =vacio **then**
d_1
else
if ¬def?(dameUno(claves(d_0)),d_1)∨$_L$ obtener(dameUno(claves(d_0)), d_0)> obtener(dameUno(claves(d_0)), d_1) **then**
unirDicc(borrar(dameUno(claves(d_0)), d_0), definir(dameUno(claves(d_0)),obtener(dameUno(claves(d_0)),d_0), d_1))
else
unirDicc(borrar(dameUno(claves(d_0)), d_0), definir(dameUno(claves(d_0)),obtener(dameUno(claves(d_0)),d_1), d_1))
fi
fi
incrementar(c,d) ≡ **if** d=vacio **then**
d
else
incrementar(sinUno(c), definir(dameUno(c) , obtener(dameUno(c))+1 , d))
fi

aDistancia3(Pos) $\equiv$ {(Π_1(Pos)+3,Π_2(Pos)),(Π_1(Pos)-3,Π_2(Pos)), (Π_1(Pos),Π_2(Pos)+3), (Π_1(Pos),Π_2(Pos)-3), (Π_1(Pos)+3,Π_2(Pos)), (Π_1(Pos)+2,Π_2(Pos)-1), (Π_1(Pos)+2,Π_2(Pos)+1), (Π_1(Pos)-2,Π_2(Pos)-1), (Π_1(Pos)-2,Π_2(Pos)+1), (Π_1(Pos)+1,Π_2(Pos)-2), (Π_1(Pos)-1,Π_2(Pos)-2), (Π_1(Pos)+1,Π_2(Pos)+2), (Π_1(Pos)-1,Π_2(Pos)+2) }

corregComercios(c,d_{com},cas) $\equiv$ **if** c=vacio **then**
d_{com}
else
corregComercios(sinUno(c), definir(dameUno(c), nivelMaximo(0, aDistancia3(dameUno(c)) , d_{com}) , cas))
fi

construcciones(s) $\equiv$ claves(casas(s))$\cup$claves(comercios(s))

posOcupada(s,p) $\equiv$ p $\in$ construcciones(s) $\vee$ hayRioEnPos(mapa(s), {p})

Fin TAD

TAD Nombre *es* String

1.3. TAD Servidor

TAD SERVIDOR

igualdad observacional

$$(\forall s, s' : \text{SimCity}) \left(s =_{\text{obs}} s' \iff \begin{pmatrix} \text{partidas}(s) =_{\text{obs}} \text{partidas}(s') \wedge_{\text{L}} \\ (\forall n : \text{Nombre})(\text{def?(n,partidas(s))} \rightarrow_{L} \\ \text{fueUnion(partida(s,n)} =_{\text{obs}} \text{fueUnion(partida(s',n))} \end{pmatrix} \right)$$

géneros Servidor

exporta Servidor, observadores, generadores, turnos, partida, sePuedeUnir, agregoAlgo

usa SimCity

observadores básicos

partidas	: servidor	$\longrightarrow$ dicc(Nombre, Simcity)	
fueUnion	: servidor s $\times$ Nombre n	$\longrightarrow$ bool	{def?(n, partidas(s))}

generadores

iniciarServidor	:	$\longrightarrow$ Servidor	
iniciarPartida	: Servidor s $\times$ Nombre n $\times$ Mapa m	$\longrightarrow$ Servidor	{$\neg$ def?(n,partidas(s))}
AgCasa	: Servidor s $\times$ Nombre n $\times$ Pos p	$\longrightarrow$ Servidor	{def?$(n, partidas(s)) \wedge_{\text{L}} \neg$ fueUnion$(n, s) \wedge \neg$ posOcupada(partida(s,m),p)}
AgComercio	: Servidor s $\times$ Nombre n $\times$ Pos p	$\longrightarrow$ Servidor	{def?$(n, partidas(s)) \wedge_{\text{L}} \neg$ fueUnion$(n, s) \wedge \neg$ posOcupada(partida(s,m),p)}
avanzarTurno	: Servidor s $\times$ Nombre n	$\longrightarrow$ Servidor	{$def?(n, partidas(s)) \wedge_{\text{L}} \neg$ fueUnion(s) $\wedge$ agregoAlgo(partida(s,n))}
unir	: Servidor s $\times$ Nombre a $\times$ Nombre b	$\longrightarrow$ Servidor	$\left\{ \begin{array}{l} \text{def?(a,partidas(s))} \wedge \text{def?(b,partidas(s))} \wedge_{L} \neg \text{fueUnion(b)} \wedge \\ \neg\text{hayRioEnPos(mapa(partida(s,a)),construcciones(partida(s,b))} \wedge \\ \neg\text{hayRioEnPos(mapa(partida(s,b)), construcciones(partida(s,a))} \wedge \\ (\forall \text{c:Construccion})(\text{c}\in(\text{construcciones(partida(s,a))} \cap \text{construcciones(partida(s,b))} \rightarrow_{L} \\ \text{nivel(c,partida(s,a))} < \text{turno(partida(s,a))} \wedge \\ \text{nivel(c, partida(s,b))}) < \text{turno(partida(s,b))} \end{array} \right\}$

otras operaciones

partida	: Servidor s $\times$ Nombre n	$\longrightarrow$ SimCity	{def?(n,partidas(s))}

axiomas

partidas(iniciarServidor)	$\equiv$	vacio
partidas(inicipartidaPartida(s,n,m))	$\equiv$	definir(n , iniciar(m) , partidas(s))
partidas(AgCasa(s,n,p))	$\equiv$	definir(n , agregarCasaSC(partida(s,n),p) , partidas(s))
partidas(AgComercio(s,n,p))	$\equiv$	definir(n , agregarComSC(partida(s,n),p) , partidas(s))
partidas(avanzarTurno(s,n))	$\equiv$	definir(n, avanzarTurnoSC(partida(s,n)), partidas(s))
partidas(unir(s,a,b))	$\equiv$	definir(a, unirSC(partida(s,a),partida(s,b)), partidas(s))
fueUnion(iniciarPartida(s,n,m),n_2)	$\equiv$	**if** $n = n_2$ **then** *false* **else** $fueUnion(s, n)$ **fi**
fueUnion(AgCasa(s,n,p),n_2)	$\equiv$	fueUnion(s,n_2)
fueUnion(AgComercio(s,n,p),n_2)	$\equiv$	fueUnion(s,n_2)
fueUnion(avanzarTurno(s,n),n_2)	$\equiv$	fueUnion(s,n_2)
fueUnion(unir(s,a,b),n)	$\equiv$	**if** $n = b$ **then** *true* **else** $fueUnion(s, n)$ **fi**
partida(s,n)	$\equiv$	obtener(n,partidas(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: `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

VERTICALES(**in** m: `mapa`) $\rightarrow$ res : `conj(Nat)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} verticales(m)\}$
Complejidad: $O(1)$
Descripción: Devuelve las posiciones donde estan los ejes de los rios verticales
Aliasing: El resultado es una referencia modificable al conjunto almacenado

HORIZONTALES(**in** m: `mapa`) $\rightarrow$ res : `conj(Nat)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} horizontales(m)\}$
Complejidad: $O(1)$
Descripción: Devuelve las posiciones donde estan los ejes de los rios verticales
Aliasing: El resultado es una referencia modificable al conjunto almacenado

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 $\Longleftrightarrow$ 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: $estr.verticales \leftarrow vs$
3: **return** estr

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

---

iHorizontales(**in** e : `estr`) $\rightarrow$ res : conj(Nat)

1: $res \leftarrow e.horizontales$
2: **return** res

Complejidad: $O(1)$

iVerticales(**in** e : `estr`) $\rightarrow$ res : conj(Nat)

1: $res \leftarrow e.verticales$
2: **return** res

Complejidad: $O(1)$

2.2. Módulo SimCity

Interfaz

se explica con: SimCity

géneros: `SimCity`

Operaciones básicas

MAPA(**in** s : `SimCity`) $\rightarrow$ res : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} mapa(s)\}$
Complejidad: $O(1)$
Descripción: Devuelve el mapa del SimCity
Aliasing: El resultado es una referencia no modificable al mapa almacenado en la estructura

CASAS(**in** s : `SimCity`) $\rightarrow$ res : `dicc(`pos`, `$nivel$`)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} casas(s)\}$
Complejidad: $O(1)$
Descripción: devuelve las la posición de las casas y su nivel
Aliasing: El resultado es una referencia no modificable al diccionario almacenado en la estructura

COMERCIOS(**in** s : `SimCity`) $\rightarrow$ res : `dicc(`pos`, `$nivel$`)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} comercios(s)\}$
Complejidad: $O(|casas| * |comercios|)$
Descripción: devuelve las posiciones de los comercios y su nivel
Aliasing: El resultado es una referencia no modificable al diccionario almacenado en la estructura

POPULARIDADSC(**in** s : `SimCity`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} popularidad(s)\}$
Complejidad: $O(1)$
Descripción: Devuelve la popularidad del SimCity

TURNOSSC(**in** s : `SimCity`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} turnos(s)\}$
Complejidad: $O(1)$
Descripción: Devuelve el turno del SimCity

INICIARSC(**in** m : `mapa`) $\rightarrow$ res : `SimCity`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} iniciar(m)\}$
Complejidad: $O(1)$
Descripción: Crea SimCity a partir de un mapa

AVANZARTURNOSC(**in/out** s : `SimCity`)
Pre $\equiv$ $\{s = s_0 \wedge$ agregoAlgo(s)$\}$
Post $\equiv$ $\{s =_{\text{obs}} avanzarTurno(s_0, d)\}$
Complejidad: $O(|casas| + |comercios|)$
Descripción: Avanza el turno de la partida, actualizando el nivel de los comercios y las casas

AGREGARCASASC(**in/out** s: `SimCity`, **in** p: `Pos`)
Pre $\equiv \{s = s_0 \wedge \neg posOcupada(s, p)\}$
Post $\equiv \{s =_{\text{obs}} agregarCasaSC(s_0, p)\}$
Complejidad: $O(1)$
Descripción: Agrega una casa a SimCity con nivel 0

AGREGARCOMERCIOSC(**in/out** s: `SimCity`, **in** p: `Pos`)
Pre $\equiv \{s = s_0 \wedge \neg posOcupada(s, p)\}$
Post $\equiv \{s =_{\text{obs}} agregarComSC(s_0, p)\}$
Complejidad: $O(1)$)
Descripción: Agrega un comercio a SimCity con nivel 0

UNIRSC(**in/out** a: `SimCity`,**in** b: `SimCity`)
Pre $\equiv \{a = a_0 \wedge \neg$hayRioEnPos(construcciones(b),mapa(a)) $\wedge$
$(\forall$c:Construccion$)($c$\in$(construcciones(a) $\cap$ construcciones(b))$\rightarrow_L$
nivel(c,a) $<$ turno(a) $\wedge$ nivel(c,b) $<$ tuno(b) $\}$
Post $\equiv \{a =_{\text{obs}} unir(a_0, b)\}$
Complejidad: $O(|hsA| * |hsB| + |vsA| * |vsB| + |casasA \cup casasB| * |comerciosA \cup comerciosB|)$
Descripción: Actualiza el simCity a, como la union con el simCity b

Operaciones auxiliares (no se exportan):

SUMAR1ATODOS(**in/out** d: `dicc(Pos, nivel)`)
Pre $\equiv \{s =_{\text{obs}} s_0\}$
Post $\equiv \{$claves(s) $=_{\text{obs}}$ claves(s$_0$)$\wedge_{\text{L}}(\forall p$: Posicion$)($def?(p,d) $\rightarrow_L$ obtener$(p, s) =$ obtener$(p, s_0) + 1)\}$
Complejidad: $O(|d|)$
Descripción: Incrementa el nivel de todas las construcciones en 1

ACTUALIZARCOMERCIOS(**in** *casas*: `dicc(Pos,Nivel)`, **in/out** *comercios*: `dicc(Pos,Nivel)`)
Pre $\equiv \{comercios = comercios_0\}$
Post $\equiv \{comercios =_{\text{obs}} corregComercios(claves(comercios_0), comercios_0, casas)\}$
Complejidad: $O(|$comercios$|*|$casas$|))$
Descripción: Agrega un comercio a SimCity con nivel 0

DISTANCIAMANHATTAN(**in** $Pos1$: `Posicion`, **in** $Pos2$: `Posicion`)
Pre $\equiv \{$true$\}$
Post $\equiv \{res = |\pi_0(Pos1) - \pi_0(Pos2)| + |\pi_1(Pos1) - \pi_1(Pos2)|\}$
Complejidad: $O(1)$
Descripción: Calcula la distancia de Manhattan entre dos posiciones

Representación

Representación de SimCity

Un SimCity se representa como una tupla de 5 valores donde cada posición representa a un observador. Los observadores casas y comercios se representan como diccionarios lineales, de modo de agregar un nuevo valor en O(1) mediante la operación de definir rápido. En el caso de los comercios, la complejidad es lineal con respecto a la cantidad de casas, pues hay que mirar si hace falta una actualización de nivel debido a una distancia de Manhattan con una casa menor a 3.

Los conflictos de la union se resuelven descartando la construcción de menor nivel (sin importar si es casa o comercio). En caso de igual nivel, se descarta la construcción del simCity que se esta uniendo. Por precondicion, se protegen aquellos casos donde el conflicto ocurre en una construcción de nivel maximo

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

donde `estr` es `tupla(`*mapa*: `mapa` , *turno*: `Nat` , *popularidad*: `Nat` , *agregoEnTurno*: `bool`
, *casas*: `diccLineal(`*pos*`: Posición,` *nivel*`: nat)`
, *comercios*: `diccLineal(`*pos*`: Posición,` *nivel*`: nat) )`

Invariante de representación

Para mantener la validez de nuestra estructura necesitamos las siguientes condiciones:

1- No puede haber una casa y un comercio en una misma posición, es decir, la intersección entre las claves de ambos diccionarios debe ser nula.

2- En todas las posiciones donde hay un rio, no puede haber ni una casa, ni un comercio.
3- Ningun nivel de una construccion puede ser superior al turno actual en que nos encontramos

$(\forall e : estr)$

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\iff$ (claves(e.casas) $\cap$ claves(e.comercions) $= \emptyset$)$\wedge$
$(\forall p : \text{Pos})(p \in (\text{claves(e.casas)} \cup \text{claves(e.comercions)}) \rightarrow_L \neg\text{hayRioEnPos(mapa(e), p)}) \wedge$
$(\forall p : \text{Pos})(p \in (\text{claves(e.casas)} \rightarrow_L \text{obtener(p,e.casas)} \leq \text{e.turno}) \wedge$
$(\forall p : \text{Pos})(p \in (\text{claves(e.comercios)} \rightarrow_L \text{obtener(p,e.comercios)} \leq \text{e.turno}) \wedge$

Funcion de abstacción

$(\forall e : estr)(\forall s : servidor)$

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

Abs(e) $\equiv$ s | e.mapa $=_{\text{obs}}$ mapa(s) $\wedge$ e.turno $=_{\text{obs}}$ turno(s) $\wedge$ e.popularidad = popularidad(s) $\wedge$
e.casas $=_{\text{obs}}$ casas(s) $\wedge$ corregComercios(e.comercios) $=_{\text{obs}}$ comercios(s)

Algoritmos

---

iIniciarSC(**in** m : `mapa`) $\rightarrow$ res : estr

1: $res \leftarrow < m, 0, 0, false, Vacio(), Vacio() >$
2: **return** res

Complejidad: $O(1)$

---

iCasas(**in** e : `estr`) $\rightarrow$ res : diccLineal(pos: Posicion, $nivel$: nat)

1: $res \leftarrow e.casas$
2: **return** res

Complejidad: $O(1)$

---

iComercios(**in** e : `estr`) $\rightarrow$ res : diccLineal(pos: Posicion, $nivel$: nat)

1: $res \leftarrow actualizarComercios(e.casas, e.comercios)$
2: **return** res

Complejidad: $O(|casas| * |comercios|)$

---

iPopularidadSC(**in** e : `estr`) $\rightarrow$ res : Nat

1: $res \leftarrow e.popularidad$
2: **return** res

Complejidad: $O(1)$

---

iTurnosSC(**in** e : `estr`) $\rightarrow$ res : Nat

1: $res \leftarrow e.turnos$
2: **return** res

Complejidad: $O(1)$

---

iAvanzarTurnoSC(**in**/**out** e : `estr`)

1: $sumar1Atodos(e.casas)$ ▷ $O(|casas|)$
2: $sumar1Atodos(e.comercio)$ ▷ $O(|comercios|)$
3: $e.turno++$ ▷ $O(1)$

Complejidad: $O(|casas| + |comercios|)$

iAgregarCasaSC(**in**/**out** e : `estr`, **in** p : `Posicion`)

1: $DefinirRapido(e.casas, p, 0)$

Complejidad: $O(1)$

iAgregarComercioSC(**in**/**out** e : `estr`, **in** p : `Posicion`)

1: $DefinirRapido(e.comercios, p, n)$

Complejidad: $O(1))$

iActualizarComerciosSC(**in** *casas* : `dicc(Pos, Nivel)` , **in**/**out** *comercios* : `dicc(Pos,Nivel)`)

1: $itComercios \leftarrow crearIt(comercios)$
2: **while** haySiguiente($itComercios$)) **do**
3: $p \leftarrow SiguienteClave(itComercios)$
4: $n \leftarrow SiguienteSignificado(itComercios)$ ▷ Obtenemos una referencia modificable
5: $itCasas \leftarrow crearIt(casas)$
6: **while** haySiguiente($itCasas$) **do** ▷ $O(|casas|)$
7: $posAct \leftarrow SiguienteClave(itCasas)$
8: $nivelAct \leftarrow SiguienteSignificado(itCasas)$
9: **if** distanciaDeManhattan$(p, posAct) = 3 \wedge nivelAct > n$ **then**
10: $n = nivelAct$
11: **end if**
12: $Avanzar(itCasas)$
13: **end while**
14: $Avanzar(itComercios)$
15: **end while**

Complejidad: $O(|casas| * |comercios|))$

sumar1Atodos(**in**/**out** d : `diccLineal(Pos, nivel)`)

1: $itDicc \leftarrow crearIt(itDicc)$
2: **while** haySiguiente(itDicc) **do**
3: $nivelAct \leftarrow SiguienteSignificado(itDicc)$
4: $nivelAct++$
5: $Avanzar(itDicc)$
6: **end while**
7: **return** res

Complejidad: $O(|d|))$

iDistanciaDeManhattan(**in** $Pos1$: `Posicion`, **in** $Pos2$: `Posicion`)

1: $res \leftarrow |\pi_0(Pos1) - \pi_0(Pos2)| + |\pi_1(Pos1) - \pi_1(Pos2)|$
2: **return** res

Complejidad: $O(1)$

iUnirSC(**in/out** $e1$: `estr`, **in** $e2$: `estr`)

1: **if** e2.turnos >e1.turnos **then**
2: e1.turnos = e2.turnos
3: **end if**
4: $horizontalesE1 \leftarrow horizontales(mapa(E1))$
5: $itHorizontalesE2 \leftarrow crearIt(horizontales(mapa(E2)))$
6: **while** HaySiguiente($itHorizontalesE2$) **do** ▷ $O(|hsE1| * |hsE2|)$
7: $Agregar(horizontalesE1, Siguiente(itHorizontalesE2))$
8: avanzar(itHorizontalesE2)
9: **end while**
10: $itVerticalesE2 \leftarrow crearIt(verticales(mapa(E2)))$
11: $verticalesE1 \leftarrow verticales(mapa(E1))$
12: **while** $HaySiguiente(itVerticalesE2)$ **do** ▷ $O(|vsE1| * |vsE2|)$
13: $Agregar(verticalesE1, Siguiente(itVerticalesE2))$
14: avanzar(itVerticalesE2)
15: **end while**
16: $itCasas \leftarrow crearIt(e2.casas)$
17: **while** HaySiguiente(itCasas) **do** ▷ $O(|casasE1| * |casasE2|)$
18: **if** ¬Definido?(e1.casas, SiguienteClave(itCasas)) $\vee_L$ Significado(SiguienteClave(itCasas),e1.casas)< SiguienteSignificado(itCasas) **then**
19: Definir(e1.casas, SiguienteClave(itCasas) , SiguienteSignificado(itCasas))
20: **end if**
21: avanzar(itCasas)
22: **end while**
23: $itComercios \leftarrow crearIt(e2.comercios)$
24: **while** HaySiguiente(itComercios) **do** ▷ $O(|comerciosE1| * |comerciosE2|)$
25: **if** ¬Definido?(e1.comercios, SiguienteClave(itComercios)) $\vee_L$
26: Significado(SiguienteClave(itComercios),e1.comercios)< SiguienteSignificado(itComercios) **then**
27: Definir(e1.comercios, SiguienteClave(itComercios) , SiguienteSignificado(itComercios))
28: **end if**
29: avanzar(itComercios)
30: **end while**
31: $actualizarComerciosSC(e1)$ ▷ $O(|casasE1 \cup casasE2| * |comerciosE1 \cup comerciosE2|)$
32: $e1.popularidad = e1.popularidad + e2.popularidad + 1$
33: e1.agregoEnTurno ← e1.agregoEnTurno ∨ e2.agregoEnTurno

Complejidad: $O(|hsE1| * |hsE2| + |vsE1| * |vsE2| + |casasE1 \cup casasE2| * |comerciosE1 \cup comerciosE2|))$

2.3. Módulo Servidor

Interfaz

se explica con: SERVIDOR

géneros: `servidor`

Operaciones básicas de servidor

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

INICIARPARTIDA(**in/out** s: `servidor`, **in** $Nombre$: `String`,**in** m: `Mapa`)
Pre $\equiv$ {$s = s_0 \wedge \neg def?(Nombre, partidas(s))$}
Post $\equiv$ {$s =_{\text{obs}} iniciarPartida(s_0, Nombre, m)$}
Complejidad: $O(1)$
Descripción: agrega una nueva partida al servidor

AVANZARTURNO(**in/out** *servidor*: `s`, **in** $Nombre$: `String`)
Pre $\equiv$ {$s = s_0 \wedge def?(Nombre, partidas(s)) \wedge_{\text{L}} agregoAlgo(partida(s, Nombre))$}
Post $\equiv$ {$s =_{\text{obs}} avanzarTurno(s_0, partida(s, n)))$}
Complejidad: $O(|uniones(partida(Nombre1), s)| * \{|hs(mapa(partida(Nombre1), s))| * |hs(mapa(partida(Nombrei), s)| + |vs(mapa(partida(Nombrei), s)| * |vs(mapa(partida(Nombrei), s))| + |casas(partida(Nombre1, s)) \cup casas(partida(Nombrei, s))| * |comercios(partida(Nombre1, s) \cup comercios(partida(Nombrei, s)|\})$
Descripción: Avanza el turno de la partida seleccionada, resolviendo los conflictos de las uniones pendientes y agregando todas las construcciones que se fueron pidiendo antes de avanzar el turno

AGREGARCASA(**in/out** *servidor*: `s`, **in** $Nombre$: `String`, **in** p: `Posicion`)
Pre $\equiv$ {$s = s_0 \wedge def?(Nombre, partidas(s)) \wedge_{\text{L}} \neg fueUnion(s, Nombre) \wedge \neg posOcupada(partida(s, nombre), p)$}
Post $\equiv$ {$s =_{\text{obs}} AgCasa(s_0, Nombre)$}
Complejidad: $O(|Nombre|)$
Descripción: Agrega una orden para insertar una casa en la posicion seleccionada, una vez se avance el turno

AGREGARCOMERCIO(**in/out** *servidor*: `s`, **in** $Nombre$: `String`, **in** p: `Posicion`)
Pre $\equiv$ {$s = s_0 \wedge def?(Nombre, partidas(s)) \wedge_{\text{L}} \neg fueUnion(s, Nombre) \wedge \neg posOcupada(partida(s, nombre), p)$}
Post $\equiv$ {$s =_{\text{obs}} AgComercio(s_0, Nombre)$}
Complejidad: $O(|Nombre|)$
Descripción: Agrega una orden para insertar un comercio en la posicion seleccionada, una vez se avance el turno

PARTIDA(**in** *servidor*: `s`, **in** $Nombre$: `String`) $\rightarrow$ *res* : `SimCity`
Pre $\equiv$ {$def?(Nombre, partidas(s))$}
Post $\equiv$ {$s =_{\text{obs}} partida(s, Nombre)$}
Complejidad: $O(|uniones(partida(Nombre1), s)| * \{|hs(mapa(partida(Nombre1), s))| * |hs(mapa(partida(Nombrei), s)| + |vs(mapa(partida(Nombrei), s)| * |vs(mapa(partida(Nombrei), s))| + |casas(partida(Nombre1, s)) \cup casas(partida(Nombrei, s))| * |comercios(partida(Nombre1, s) \cup comercios(partida(Nombrei, s)|\})$
Descripción: Devuelve la partida que esta registrada con ese nombre. Desde aca se puede obtener mapa, comercios o casas si se llama a los observadores de SC desde aca
Aliasing: El resultado es una referencia no modificable a la partida almacenada en nuestra estructura

TURNO(**in** s: servidor, **in** $Nombre$: String) $\rightarrow$ res : Nat
Pre $\equiv$ $\{def?(Nombre, partidas(s))\}$
Post $\equiv$ $\{res =_{\text{obs}} turnos(Nombre, s)\}$
Complejidad: $O(1)$
Descripción: Devuelve el truno del simCity pedido

POPULARIDAD(**in** s: SimCityNat,**in** $Nombre$: String) $\rightarrow$ res : Nat
Pre $\equiv$ $\{def?(Nombre, partidas(s))\}$
Post $\equiv$ $\{res =_{\text{obs}} popularidad(s, nombre)\}$
Complejidad: $O(1)$
Descripción: Devuelve la popularidad de un determinado simCity

UNIR(**in/out** s: servidor, **in** $Nombre1$: String,**in** $Nombre2$: String)
Pre $\equiv$ $\{s = s_0 \wedge def?(Nombre1, partidas(s)) \wedge$ def?(Nombre2,partidas(s)) $\wedge_L \neg$ fueUnion(Nombre1) $\wedge$
$\neg$hayRioEnPos(mapa(partida(s,Nombre1)),construcciones(partida(s,Nombre2)) $\wedge$
$\neg$hayRioEnPos(mapa(partida(s,Nombre2)), construcciones(partida(s,Nombre1)) $\wedge$
($\forall$c:Construccion)(c$\in$(construcciones(partida(s,Nombre1)) $\cap$ construcciones(partida(s,Nombre2))$\rightarrow_L$
nivel(c,partida(s,Nombre1)) $<$ turno(partida(s,Nombre1)) $\wedge$
nivel(c, partida(s,Nombre2)))$<$ turno(partida(s,Nombre2))$\}$
Post $\equiv$ $\{s =_{\text{obs}} unir(s_0, Nombre1, Nombre2)\}$
Complejidad: $O(|Nombre|)$
Descripción: Manda un pedido de actualizacion para la union entre las partidas

Representación

Representación de Servidor

Los servidores se representan como diccionario trie, donde las claves son los nombres que identifican a cada partida. Los significados son tuplas que contienen diversos parametros que necesitamos para mantener la funcionalidad. El primero es logicamente el propio SimCity asociada a ese nombre.

Tenemos ademas una lista con los nombres de los SimCities que se unieron al actual. Esto es necesario para lograr la union en O(1), puesto que se guarda la union para la actualizacion de la partida.

La estructura cuenta con un contador de popularidad, que esta para salvar el caso en que se pida la popularidad de una partida antes de un llamado a actualizarPartida. Al realizar la actualizacion, el contador se resetea, puesto que la popularidad que corresponde pasara a estar almacena en la propia instancia del simCity. Todo esto esta porque, si se unio un simCity que a su vez tuvo uniones previas, consideramos que la popularidad debe reflejar esas uniones anteriores de la partida que se le unio

En un turno se pueden agregar tantas construcciones como desee. El turno se podra avanzar si se ingreso al menos una construccion. El parametro $fueUnion$ esta para asegurar esto

Las complejidades se mantienen, porque ninguna de las operaciones con restricciones de complejidad tienen acceso directo para actualizar los campos del simCity mas alla de las inserciones a casas y comercios en O(1). Para las actualizaciones de la estructura simCity que requieren complejidad lineal o superior (actualizacion por distancia de Manhattan y uniones donde hay que resolver los conflictos de fusion), todos se sobrecargan sobre la funcion actualizarPartida.

Esta funcion se llama cada vez que se avanza un turno, o si llega a solicitar la partida desde un observador (para que la informacion que se le devuelve al usuario sea la correcta). Ninguna de estas dos operaciones lleva restriccion de complejidad asociada, y son lo que permite la baja complejidad en todo el resto.

`servidor` **se representa con** `estr`

donde *estr*: `diccTrie(`*nombre*`:String,`
`tupla(`*partida*`:SimCity,`*fueUnion*`: bool,` *uniones*`: lista(Nombre),` *popAcum*`: nat)`

Invariante de Representacion

Para mantener la validez de nuestra estructura requerimos asegurar las siguientes condiciones:

1- Cuando un SimCity aparece en alguna lista de uniones, el nombre de esa partida tiene que estar definido en nuestro dicionario, y esa entrada en el diccionario debe tener el campo $fueUnion$ activo. La implementacion luego asegura que es imposible revertir el cambio

2- El campo *popAcum* debe ser siempre la suma de las popularidades de las partidas que se encuentran en la lista de uniones para la partida actual

3-La lista de uniones no puede tener repetidos

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\iff$ $(\forall n, m : Nombre)(def?(n, e) \wedge_{\text{L}} esta?(m, significado(n, e).uniones) \rightarrow_L$
sinRepetidos(e.uniones) $\wedge$ def?(m, e.partidas) $\wedge$ obtener(m,e.partidas).fueUnion) $\wedge$
$(\forall n$: Nombre)(def?(n, e) $\rightarrow_L$ $obtener(n, e).popAcum =$
$\sum_{i=0}^{longitud(obtener(s,n).uniones)} popularidad(obtener(e, iEsimo(obtener(s, n).uniones), i))$

Funcion de abstraccion

Vamos a hacerla mediante funciones auxiliares que sean generadores. En cualquier momento dado, podria pasar que una partida en nuestra estructura tenga actualizacion pendiente, por lo que para obtener la informacion actualizada, deberiamos efectuar esa actualizacion. Nuestra funcion de abstraccion hace precisamente esas actualizaciones, pero en el mundo TADs

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

Abs(e) $\equiv$ actualizarPartidas(e, claves(e))

Funciones auxiliares de Abs

actualizarPartidas : estr e $\times$ conj(Nombres) cn $\longrightarrow$ servidor {cn $\subseteq$ claves(e)}

actualizarSC : estr e $\times$ Nombre n $\longrightarrow$ servidor {n $\in$ claves(e)}

todasLasUniones : estr e $\times$ Nombre n $\times$ lista(Nombre) su $\longrightarrow$ servidor
{n $\in$ claves(e) $\wedge$ ($\forall i$: nat)(i<longitud(obtener(n,e).uniones) $\rightarrow_L$ def?(obtener(n,e).uniones[i], i)}

efectuarUnion : estr e $\times$ Nombre a $\times$ Nombre b $\longrightarrow$ servidor {def?(a,n) $\wedge$ def?(b,n)}

```
actualizarPartidas(e, cn)   ≡ if ∅?(cn) then
                                  e
                              else
                                  actualizarPartidas(actualizarSC(e, dameUno(cn)), sinUno(cn))
                              fi
actualizarSC(e,n)           ≡ definir(todasLasUniones(e, n, obtener(s,n).uniones), e.partidas)
todasLasUniones(e,a,su)     ≡ if vacia?(su) then
                                  s
                              else
                                  todasLasUniones(efectuarUnion(a, prim(su)), a, fin(su))
                              fi
efectuarUnion(e,a,b)        ≡ definir(a,unirSC(obtener(a, e).partida), obtener(b, e).partida), e)
```

Algoritmos

iIniciarServidor() $\rightarrow$ res : estr

1: $res \leftarrow < Vacio() >$ ▷ $O(1)$
2: **return** res

Complejidad: $O(1)$

iIniciarPartida(**in/out** e : `estr`, **in** $Nombre$: `String`, **in** m : `Mapa`)

1: Definir(e.partidas, Nombre, iniciarSC(m), Vacio()) ▷ $O(|Nombre|)$

Complejidad: $O(|Nombre|)$

iAgregarCasa(**in/out** e : `estr`, **in** $Nombre$: `String`, **in** p : `Posicion`)

1: $partida \leftarrow significado(e, Nombre).partida$ ▷ $O(|Nombre|)$
2: $agregarCasaSC(partida, p)$ ▷ $O(1)$

Complejidad: $O(|Nombre|)$

iAgregarComercio(**in/out** e: `estr`, **in** $Nombre$: `String`, **in** p: `Posicion`)

1: $partida \leftarrow significado(e, Nombre).partida$ ▷ $O(|Nombre|)$
2: $agregarComercioSC(partida, p)$ ▷ $O(1)$

Complejidad: $O(|Nombre|)$

iAvanzarTurno(**in/out** e: `estr`, **in** $Nombre$: `String`)

1: $SC \leftarrow significado(e, Nombre)$ ▷ $O(|Nombre|)$
2: $actualizarPartida(SC)$
3: $avanzarTurnoSC(partida)$

Complejidad: $O(|unionesE1| * \{|hsE1| * |hsE2| + |vsE1| * |vsE2| + |casasE1 \cup casasE2| * |comerciosE1 \cup comerciosE2|\})$

iActualizarPartida(**in/out** e: `estr`, **in** $Nombre$: `String`)

1: $partida \leftarrow significado(e, Nombre).partida$ ▷ $O(|Nombre|)$
2: $itUniones \leftarrow CrearIt(s.uniones)$ ▷ $O(1)$
3: **while** HaySiguiente($itUniones$) **do** ▷ $O(|uniones)|)$
4: $unionAct \leftarrow significado(e, Siguiente(itUniones)).partida$
5: $unirSC(partida, unionAct)$
6: $EliminarSiguiente(itUniones)$
7: **end while**
8: $e.popAcum = 0$
9: $actualizarComerciosSC(partida)$ ▷ $O(|casas| * |comercios|)$
10: **return**

Complejidad: $[O(|unionesE1| * \{|hsE1| * |hsE2| + |vsE1| * |vsE2| + |casasE1 \cup casasE2| * |comerciosE1 \cup comerciosE2|\})$

iPartida(**in** e: `estr`, **in** $Nombre$: `String`)→ res : SimCity

1: actualizarPartida(e,Nombre)
2: res ← Significado(e,Nombre).partida
3: **return** res

Complejidad: $O(|unionesE1| * \{|hsE1| * |hsE2| + |vsE1| * |vsE2| + |casasE1 \cup casasE2| * |comerciosE1 \cup comerciosE2|\})$

iUnir(**in/out** e: `estr`, **in** $Nombre1$: `String`, **in** $Nombre2$: `String`)

1: $sc1 \leftarrow significado(e, Nombre1).partida$ ▷ $O(|Nombre|)$
2: $AgregarAtras(sc1.uniones, Nombre2)$ ▷ $O(1)$
3: $sc2 \leftarrow significado(e, Nombre2).partida$ ▷ $O(|Nombre|)$
4: $AgregarAtras(sc1.uniones, Nombre2)$ ▷ $O(1)$
5: $sc2.union = true$ ▷ $O(1)$
6: $sc1.popAcum = sc1.popAcum + popularidad(sc2.partida) + 1$ ▷ $O(1)$

Complejidad: $O(|Nombre|)$

iTurno(**in** $Nombre$: `String`, **in** e: `estr`) → res : Nat

1: $partida \leftarrow significado(e, Nombre).partida$ ▷ $O(|Nombre|)$
2: $res \leftarrow turnoSC(partida)$ ▷ $O(1)$
3: **return** res

Complejidad: $O(|Nombre|)$

iPopularidad(**in** e : `estr`, **in** $Nombre$: `String`) → res : Nat

1: $juego \leftarrow significado(e, Nombre)$ ▷ $O(|Nombre|)$
2: $res \leftarrow popularidadSC(juego.partida) + juego.popAcum$ ▷ $O(1)$
3: **return** res

Complejidad: $O(|Nombre|)$

Cotas de complejidad

Para hacer mas facil la lectura (y para que entre en un solo renglon), simplificamos como escribimos lo que representa la complejidad en cada caso. A continuacion se detalla a que nos referimos en cada caso:

Para todos los diccionarios d, $O(|d|)$ es una abreviacion para $O(\#claves(d))$

Cuando lleva un E1, o E2 asociado, hacemos referencia a que nos estamos refiriendo a las casas o comercios de esa estructura, por ejemplo $O(|casasE1|) = O(\#claves(casas(e1)))$ Para un conjunto hs, $O(|hs|)$ es una abreviacion para $O(longitud(hs)$

Para una lista l, $O(|l|)$ es una abreviacion para $O(longitud(l))$

2.4. Módulo diccTrie(α, β)

Interfaz

se explica con: Diccionario

géneros: `diccTrie`

Operaciones básicas de diccionario

Vacio() → res : `diccTrie`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} vacio$}
Complejidad: $O(1)$
Descripción: crea un diccionario vacio

Significado(**in** d : `diccTrie`, **in** a :α :) → res : β
Pre ≡ {$def?(a, d)$}
Post ≡ {$res =_{\text{obs}} obtener(a, d)$}
Complejidad: $O(|Nombre|)$
Descripción: Devuelve el significado de a en el diccionario d. $|Nombre|$ es la longitud de la clave más larga
Aliasing: El resultado es una referencia modificable al elemento dentro del diccionario

Definir(**in**/**out** d : $diccTrie$: , **in** a :α : , **in**b :β :)
Pre ≡ {$d = d_0$}
Post ≡ {$d =_{\text{obs}} definir(a, d_0)$}
Complejidad: $O(|Nombre|)$
Descripción: Define la clave a con el valor b. $|Nombre|$ es la longitud de la clave más larga. Tanto a como b se agregan por referencia