DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Diseño SimCity

Integrante	LU	Correo electrónico
Sosa, Patricio	218/16	patriciososa91@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1. Módulo Mapa

Interfaz

se explica con: MAPA

géneros: `mapa`

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

HAYRIO?(**in** m : `mapa`,**in** c : `casilla`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} hayRio(m, c)\}$
Complejidad: $O(m.horizontales)$
Descripción: verifica si una casilla es un rio

AGREGARRIO(**in/out** m : `mapa`,**in** d : `direccion`,**in** p : `posicion`)
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} agregarRio(m, d, p)\}$
Complejidad: $O(1)$
Descripción: agrega un rio

UNIRMAPA(**in** $m1$: `mapa`,**in** $m2$: `mapa`) $\rightarrow$ res : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} unirMapa(m1, m2)\}$
Complejidad: $O(m2.horizontales + m2.verticales)$
Descripción: une dos mapas

Representación

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 $\{\text{Rep}(m)\}$
Abs(m) $\equiv$ c /

(∀cas: casilla) hayRio(c,cas) $\iff$ π1(cas) $\in$ m.horizontales $\vee$ π2(cas) $\in$ m.verticales

Algoritmos

---

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

1: $estr.horizontales \leftarrow hs$ ▷ O(copy(hs))
2: $estr.verticales \leftarrow vs$ ▷ O(copy(vs))
3: **return** estr

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

---

ihayRio?(**in** m : `mapa`, **in** c : `casilla`) $\rightarrow$ res : bool

```
1: itHor ← crearIt(m.horizontales)                                  ▷ O(1)
2: while (haySiguiente?(itHor)) do                                  ▷ O(1)
3:     if (c.first == Siguiente(itHor)) then                        ▷ O(1)
4:         return true                                              ▷ O(1)
5:     end if
6:     Avanzar(itHor)                                               ▷ O(1)
7: end while
8: itVer ← crearIt(m.verticales)                                    ▷ O(1)
9: while (haySiguiente?(itVer)) do                                  ▷ O(1)
10:    if (c.second == Siguiente(itVer)) then                       ▷ O(1)
11:        return true                                              ▷ O(1)
12:    end if
13:    Avanzar(itVer)                                               ▷ O(1)
14: end while
15: return false                                                    ▷ O(1)
```

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

---

iagregarRio(**in/out** m : `mapa`, **in** d : `direccion`,**in** p : `posicion`)

```
1: if d == Horizontal then
2:     agregarRapido(m.horizontales,p)
3: else
4:     agregarRapido(m.verticales,p)
5: end if
```

Complejidad: $O(1)$

---

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

```
1: itHor ← crearIt(m2.verticales)
2: itVer ← crearIt(m2.horizontales)
3: while (haySiguiente?(itHor)) do
4:     agregar(m1.horizontales, Siguiente(itHor))
5:     Avanzar(itHor)
6: end while
7: while haySiguiente?(itVer) do
8:     agregar(m1.verticales, Siguiente(itVer))
9:     Avanzar(itVer)
10: end while
```

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

2. Módulo SimCity

Interfaz

se explica con: SimCity

géneros: sc

imapa(**in** s : SimCity) $\rightarrow$ res : mapa
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} mapa(s)\}$
Complejidad: $O((s.mapasRecienttesDeOtrosSC) * unirMapa)$
Descripción: muestra el mapa el SimCity

icasas(**in** s : simCity) $\rightarrow$ res : conj(Casilla)
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} casas(s)\}$
Complejidad: $O(iUnirCasasPriorizandoMaximoNivel(s))$
Descripción: Muestra las casas que contiene el SimCity

icomercios(**in** s : SimCity) $\rightarrow$ res : conj(Casilla)
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} comercios(s)\}$
Descripción: muestra los comercios que contiene el SimCity

inivel(**in** s : simCity, **in** c : casilla) $\rightarrow$ res : nat
Pre $\equiv$ {hayConstruccion?(s,c)}
Post $\equiv$ $\{res =_{\text{obs}} nivel(s, c)\}$
Descripción: Dado un simCity y una construccion me devuelve el nivel de esa construccion

ihuboConstruccion(**in** s : SimCity) $\rightarrow$ res : bool
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} huboConstruccion(s)\}$
Complejidad: $O(1)$
Descripción: pregunta si se hizo una construccion en ese turno

iturno(**in** s : SimCity) $\rightarrow$ res : nat
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} antiguedad(s)\}$
Complejidad: $O(1)$
Descripción: Devuelve el turno actual

ipopularidad(**in** s : simCity) $\rightarrow$ res : nat
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} pupularidad(s)\}$
Complejidad: $O(1)$
Descripción: Devuelve la popularidadde la partida, es decir la cantidad de partidas que se unieron

iempezarPartida(**in** m : mapa) $\rightarrow$ res : SimCity
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} empezarPartida(m)\}$
Complejidad: $O(1)$
Descripción: empieza una partida de SimCity

iAgregarCasa(**in/out** s : SimCity, **in** c : casilla)
Pre $\equiv$ {sePuedeConstruir(s,c)}
Post $\equiv$ $\{res =_{\text{obs}} agregarCasa(s, c)\}$
Complejidad: $O(1)$
Descripción: Dado un simCity y un casillero, me agrega una casa a ese casillero en caso de que se pueda construir

iagregarComercio(**in/out** s : SimCity,**in** c : casilla)

Pre ≡ {sePuedeConstruir(s,c)}
Post ≡ $\{res =_{\text{obs}} agregarComercio(s, c)\}$
Complejidad: $O(1)$
Descripción: Dado un simCity y un casillero, me agrega un comercio a ese casillero en caso de que se pueda construir

IAVANZARTURNO(**in/out** s : `simCity`)
Pre ≡ {huboConstruccion?(s)}
Post ≡ $\{res =_{\text{obs}} avanzarTurno(s)\}$
Descripción: Dado un simCity, en caso de que hubo una construcción en el turno actual, puedo avanzar al siguiente turno

IUNIR(**in/out** $s1$: `simCity`, **in** $s2$: `simCity`)
Pre ≡ {(∀ c: casilla)(c ∈ construcciones(s1) $\longrightarrow_L$ hayRio(mapa(s2),c)) $\wedge_L$ (∀ c : casilla)(c ∈ construcciones(s1) $\longrightarrow_L$ hayRio(mapa(s1),c)) $\wedge_L$ (∀ c1, c2 : casilla)(c1 ∈ construcciones(s1) ∧ c2 ∈ construcciones(s2)$\longrightarrow$ (esCasillaDeMaximoNivel(s1,c1)) ∧ esCasillaDeMaximoNivel(s2,c2) $\longrightarrow$ c1 ≠ c2)}
Post ≡ $\{res =_{\text{obs}} unir(s1, s2)\}$
Complejidad: $O(1)$
Descripción: dado dos SimCity, los une en uno nuevo

ICONSTRUCCIONES(**in** s : `simCity`) → res : `conj(Casilla)`
Pre ≡ {true}
Post ≡ $\{res =_{\text{obs}} construcciones(s)\}$
Complejidad: $O(\#claves(comercios(s))$
Descripción: devuelve las construcciones de un SimCity

IHAYCONSTRUCCION(**in** s : `simCity`,**in** c : `Casilla`) → res : `bool`
Pre ≡ {true}
Post ≡ $\{res =_{\text{obs}} hayCostruccion(s, c)\}$
Complejidad: $O(\#construcciones(s))$
Descripción: Dado un simCity y una casilla, devuelve si en dicha casilla hay una construccion

ISEPUEDECONSTRUIR(**in** s : `simCity`,**in** c : `Casilla`) → res : `bool`
Pre ≡ {true}
Post ≡ $\{res =_{\text{obs}} sePuedeConstruir(s, c)\}$
Complejidad: $O(hayRio(mapa(s), c) + hayConstrucciones(s, c))$
Descripción: Dado un simCity y una casilla, devuelve si en dicha casilla se puede construir una construccion

IESCASILLADEMAXIMONIVEL(**in** s : `simCity`,**in** c : `Casilla`) → res : `bool`
Pre ≡ {hayConstrucciones(s,c)}
Post ≡ $\{res =_{\text{obs}} esCasillaDeMaximoNivel(s, c)\}$
Complejidad: Dado un simCity y una casilla, devuelve si en dicha casilla es una construccion de maximo nivel

IUNIRCASASPRIORIZANDONIVELMAXIMO(**in** s : `simCity`) → res : `diccLineal(casilla,nat)`
Pre ≡ {true}
Post ≡ $\{claves(res) =_{\text{obs}} unirCasasPriorizandoNivelMaximo(s, empezarPartido(crear()))\}$
Complejidad: $O((2*\#s.casasYComerciosRecientesDeOtroSC*\#s.casas^2)+(3*\#s.CasasYComerciosRecientesDeOtroS$
$\#s.casas))$
Descripción: Dado un simcity devuelve las casas priorizando el maximo nivel.

IUNIRCOMERCIOSPRIORIZANDONIVELMAXIMO(**in** s : `simCity`) → res : `diccLineal(casilla,nat)`
Pre ≡ {true}
Post ≡ $\{claves(res) =_{\text{obs}} unirComerciosPriorizandoNivelMaximo(s, empezarPartido(crear()))\}$
Complejidad: $O((2*\#s.comerciosDeOtroSC*\#s.comercios^2)+(3*\#s.comerciosDeOtroSC^2*\#s.comercios))$
Descripción: Dado un simcity me devuelve los comercios priorizando el maximo nivel

ICASASADISTANCIA3MANHATTAN(**in** cc : `conj(Casilla)`,**in** c : `Casilla`) → res : `conj(Casilla)`
Pre ≡ {true}
Post ≡ $\{res =_{\text{obs}} casasADistancia3Manhattan(cc, c)\}$
Complejidad: $O(\#cc)$
Descripción: dado un conjunto de construcciones y una casilla, devuelve las construcciones a una distancia manhattan de dicha casilla

IMAXIMONIVEL(**in** s : `simCity`,**in** c : `Conj(Casilla)`) $\rightarrow$ res : `nat`
Pre $\equiv$ {c $\in$ construcciones(s)}
Post $\equiv$ {$res =_{\text{obs}} maximoNivel(s, c)$}
Descripción: dado un SimCity y un conjunto de construcciones, devuelve el maximo nivel de todos ellos

IUNIRCASAS(**in/out** $c1$: `diccLineal(casilla,nat)`,**in** $c2$: `diccLineal(casilla,nat)`)
Pre $\equiv$ {c $\in$ casas(s1)}
Post $\equiv$ {$claves(res) =_{\text{obs}} unirCasas(empezarPartida(crear()), empezarPartida(crear()), claves(c1))$}
Complejidad: $O(\#c1 * (\#claves(c1) + \#claves(c2)))$

IUNIRCOMERCIOS(**in** $c1$: `diccLineal(casilla,nat)`,**in** $c2$: `diccLineal(casilla,nat)`)
Pre $\equiv$ {c $\in$ comercios(s1)}
Post $\equiv$ {$claves(res) =_{\text{obs}} unirComercio(empezarPartida(crear()), empezarPartida(crear()), claves(c1))$}
Complejidad: $O(\#c1 * (\#claves(c1) + \#claves(c2)))$

IDARNIVELACOMERCIO(**in** $s1$: `simCity`,**in** c : `Casilla`) $\rightarrow$ res : `nat`
Pre $\equiv$ {c $\in$ icomercios(s1)}
Post $\equiv$ {$res =_{\text{obs}} darNivelAComercio(s1, empezarPartida(crear()))$}
Complejidad: $O(\#s.casasYComerciosRecientesDeOtrosSC^2)$

IMAXNIVELMANHATTAM(**in** c : `casilla`,**in** d : `diccLineal(casilla,nat)`) $\rightarrow$ res : `nat`
Pre $\equiv$ {c $\in$ claves(d)}
Post $\equiv$ {res = iMaxNivelManhattam(c,d)}
Complejidad: $O(\#claves(d))$
Descripción: Devuelve el maximo nivel encontrado en casillas a dist manhattam de c

IDISTANCIAMANHATTAN(**in** x : `Casilla`,**in** y : `casilla`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} distanciaManhattan(x, y)$}
Complejidad: $O(1)$
Descripción: Devuelve la distancia manhattam

IUNION(**in/out** $c1$: `conjLineal(Casilla)`,**in** c : `conjLineal(casilla)`)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} c1 \cup c2$}
Complejidad: $O(\#c2)$
Descripción: une dos conjuntos

IUNIONDEOTROSMAPAS(**in/out** $c1$: `conjLineal(mapa)`,**in** c : `conjLineal(mapa)`)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} c1 \cup c2$}
Complejidad: $O(\#c2)$
Descripción: Une los mapas recientes de c2, en los mapas recientes de c1

Representación

`simCity` **se representa con** `estr`

donde `estr` es `tupla`(*mapa*: `mapa`
, *turno*: `nat`
, *casas*: `diccLineal(casilla,nat)`
, *comercios*: `diccLineal(casilla,nat)`
, *comerciosRecientes*: `conjLineal(casilla)`
, *ComerciosDeOtrosSC*: `conjLineal(diccLineal(casilla,nat))`
, *CasasYComerciosRecientesDeOtrosSC*:
`conjLineal(tupla(conjLineal(casilla),diccLineal(casilla,Nat)))`
, *mapasRecientesDeOtrosSC*: `conjLineal(mapa)`
, *popularidad*: `nat`
, *huboConstruccion*: `bool`)

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\iff$ 1 $\wedge_L$ 2 $\wedge_L$ 3 $\wedge_L$ 4 $\wedge_L$ 5 $\wedge_L$ 6 $\wedge_L$ 7 $\wedge_L$ 8 $\wedge_L$ 9 $\wedge_L$ 10

1. Toda casilla que pertenece a e.casas no esta en e.comercios

2. Toda casilla que pertenece a e.comercios no esta en e.casas
3. Toda casilla que pertenece a e.comerciosRecientes no esta en e.comercios ni en e.casas
4. Todos los significados de los diccionarios e.casas, e.comercios, los elementos de e.comerciosDeOtrosSC y $\pi 2$ de los elementos de e.CasasYComerciosRecientesDeOtrosSC, tienen que ser menor o igual a e.turno
5. Si hay algún comercio en e.comerciosRecientes, entonces e.huboConstruccion es true
6. Si hay alguna casa en e.casas cuyo significado sea 0, entonces e.huboConstruccion es true
7. si el cardinal de algún $\pi 1$ de cualquier elemento de e.CasasYComerciosRecientesDeOtrosSC > 0 entonces e.huboConstruccion = true
8. Toda casilla de e.casas, e.comercios, e.comerciosRecientes, e.comerciosDeOtrosSC, e.casasYComerciosRecientesDeOtrosSC no se solapan con ningún rio de e.mapa
9. e.popularidad $\geq$ # e.casasYComerciosRecientesDeOtrosSC y e.popularidad $\geq$# e.comerciosDeOtrosSC
10. # e.casasYComerciosRecientesDeOtrosSC = # e.comerciosDeOtrosSC

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

mapa(s) = e.mapa $\wedge$
casas(s) = claves(icasas(e))
comercios(s) = claves(icomercios(e)) $\wedge$

($\forall$ c: casilla)(c $\in$ construcciones(s) $\Rightarrow_{\text{L}}$nivel(s,c) = inivel(e,c))) $\wedge$
huboConstruccion(s) = e.huboConstruccion $\wedge$
popularidad(s) = e.popularidad $\wedge$
e.turno = antiguedad(s)

Algoritmos

imapa(**in** s : `SimCity`) $\rightarrow$ res : Mapa

```
1: Mapa res ← s.mapa                                          ▷ O(copy(s.mapa))
2: itAMapas ← crearIT(s.mapasRecientesDeOtrosSC)              ▷ O(1)
3: while haySiguiente(itAMapas) do                            ▷ O(1)
4:     unirMapa(res,siguiente(itAMapas))                      ▷ O(unirMapa)
5:     avanzar(itAMapas)                                      ▷ O(1)
6: end while
7: return res                                                 ▷ O(1)
```

Complejidad: $O(\#(s.mapasRecientesDeOtrosSC) * unirMapa)$

icasas(**in** s : `SimCity`) $\rightarrow$ res :conj(Casilla)

```
1: if # (s.CasasYComerciosRecientesDeOtrosSC == 0) then       ▷ O(1)
2:     return claves(s.casas)                                 ▷ O(1)
3: else
4:     dicc(casilla,nat) res ← iunirCasasPriorizandoMaximoNivel(s)   ▷ O(iunirCasasPriorizandoMaximoNivel(s))
5:
6: end if
7: return claves(res)                                         ▷ O(1)
```

Complejidad: $O(iUnirCasasPriorizandoMaximoNivel(s))$

icomercios(**in** s : `SimCity`) → res : conj(Casilla)

1: conj(casilla) resC ← s.comerciosRecientes ▷ $O(1)$
2: itComR ← crearIt(s.casasYComerciosRecientesDeOtrosSC) ▷ $O(1)$
3: **while** haySiguiente(itComR) **do** ▷ $O(\#s.casasYComerciosRecientesDeOtrosSC)$
4: resc ← iunion(resC,siguiente(itComR).first) ▷ $O(iunion())$
5: Avanzar(itComR) ▷ $O(1)$
6: **end while**
7: itRecientes ← crearIt(resC) ▷ $O(1)$
8: **while** haySiguiente(itRecientes) **do** ▷ $O(\#resC)$
9: **if** pertenece?(siguiente(itRecientes), icasas(s)) **then** ▷ $O(icasas + \#(icasas))$
10: eliminarSiguiente(itRecientes) ▷ $O(1)$
11: **end if**
12: avanzar(itRecientes) ▷ $O(1)$
13: **end while**
14: dicc(Casilla,nat) resD ← s.comercios ▷ $O(1)$
15: itConjCom ← crearIt(s.comerciosDeOtrosSC) ▷ $O(1)$
16: **while** haySiguiente(itConjCom) **do** ▷ $O(s.comerciosDeOtrosSC)$
17: resD ← iunion(iunirComercios(resD,siguiente(itConjCom)),iunirComercios(siguiente(itConjCom),resD)) ▷ $O(iUnion + iUnirComercio)$
18: **end while**
19: **if** (#s.comerciosDeOtrosSC == 0 && #resC == 0) **then** ▷ $O(1)$
20: **return** iclaves(s.comercios) ▷ $O(iclaves(s.comercios))$
21: **else**
22: **if** (#s.comerciosDeOtrosSC == 0 && #resC != 0) **then** ▷ $O(1)$
23: **return** iunion(iclaves(s.comercios),resC) ▷ $O(iunion(iclaves(s.comercios), resC))$
24: **else**
25: **if** #s.comerciosDeOtrosSC != 0 && #resC == 0 **then** ▷ $O(1)$
26: **return** iunion(iclaves(s.comercios),claves(resD)) ▷ $O(iunion(iclaves(s.comercios), claves(resD)))$
27: **else**
28: **return** iunion(iunion(iclaves(s.comercios),resC),iclaves(resD)) ▷ $O(iunion(iunion(iclaves(s.comercios), resC), iclaves(resD))$
29: **end if**
30: **end if**
31: **end if**

Complejidad: $O(\#\text{s.casasYComerciosRecientesDeOtrosSC} * iUnion())$ + $O(\$\text{resC} * O(\text{icasas}() + \#(\text{icasas}())))$ + $O(\text{s.comerciosDeOtrosSC} * (\text{iunirComercios}() + \text{iunion}()))$) +
33: max(O($iclaves(s.comercios)$),O($iunion(iclaves(s.comercios), resC)$),
34: O($iunion(iclaves(s.comercios), claves(resD))$), O($iunion$(iunion($iclaves(s.comercios)$,resC),iclaves(resD)))

inivel(**in** s : `SimCity`, **in** c : `Casilla`) → res : nat

1: **if** pertenece?(icasas(s),c) **then** ▷ $O(icasas + \#casas(s))$
2: dicc(Casilla,nat) resCas ← iunirCasasPriorizandoNivelMaximo(s) ▷ $O(iunirCasasPriorizandoNivelMaximo(s))$
3: **return** significado(resCas,c) ▷ $O(\#claves(resCas))$
4: **else**
5: dicc(Casilla,nat) resD ← iunirComerciosPriorizandoNivelMaximo(s) ▷ $O(iunirComerciosPriorizandoNivelMaximo(s))$
6: nat maximoEnPI1 ← 0 ▷ $O(1)$
7: itConj ← crearIt(s.casasYComerciosRecienAgregadosDeOtrosSC) ▷ $O(1)$
8: **while** haySiguiente(itConj) **do** ▷ $O(\#s.casasYComerciosRecienAgregadosDeOtrosSC)$
9: **if** pertenece(c,siguiente(itConj).first)) **then** ▷ $O(\#siguiente(itConj).first)$
10: **if** idarNivelAComercios(s,c) > maximoEnPI1 **then** ▷ $O(idarNivelAComercio(s,c))$
11: maximoEnPI1 ← idarNivelAComercios(s,c) ▷ $O(idarNivelAComercio(s,c))$
12: **end if**
13: **end if**
14: avanzar(itConj) ▷ $O(1)$
15: **end while**
16: **if** pertenece?(s.comerciosRecientes,c) && definido?(resD,c) **then** ▷ $O(\#s.comerciosRecientes + \#resD)$
17: **return** max{idarNivelAComercio(s,c),significado(resD,c),maximoEnPI1} ▷ $O(idarNivelAComercio(s,c) + \#resD)$
18: **else**
19: **if** pertenece?(s.comerciosRecientes,c) && !definido?(resD,c) **then** ▷ $O(\#s.comerciosRecientes + \#resD)$
20: **return** max{idarNivelAComercio(s,c),maximoEnPI1} ▷ $O(idarNivelAComercio(s,c))$
21: **else**
22: **return** max{significado(resD,C),maximoEnPI1} ▷ $O(\#resD)$
23: **end if**
24: **end if**
25: **end if**

Complejidad:O(icasas+#casas(s)) + O(iunirCasasPriorizandoNivelMaximo(s)) + O(#claves(resCas)) + O(iunirComerciosPriorizandoNivelMaximo(s)) + O(#s.casasYComerciosRecienAgregadosDeOtrosSC∗
26: $idarNivelAComercio(s,c)$ * $\#siguiente(itConj).first)$ + $O(\#s.comerciosRecientes + \#resD)$ + $O(idarNivelAComercio(s,c) + \#resD)$

---

ihuboConstruccion(**in** s : `SimCity`) → res : bool

1: **return** s.huboConstruccion ▷ O(1)

Complejidad: $O(1)$

---

iTurno(**in** s : `SimCity`) → res : nat

1: **return** s.turno ▷ O(1)

Complejidad: $O(1)$

---

ipopularidad(**in** s : `SimCity`) → res : nat

1: **return** s.popularidad ▷ O(1)

Complejidad: $O(1)$

iempezarPartida(**in** m : `mapa`) $\rightarrow$ res : SimCity

1: s.mapa $\leftarrow$ m ▷ O(1)
2: s.turno $\leftarrow$ 0 ▷ O(1)
3: s.casas $\leftarrow$ vacio() ▷ O(1)
4: s.comercios $\leftarrow$ vacio() ▷ O(1)
5: s.comerciosRecientes $\leftarrow$ vacio() ▷ O(1)
6: s.comerciosDeOtrosSC $\leftarrow$ vacio() ▷ O(1)
7: s.casasYComerciosRecientesDeOtrosSC $\leftarrow$ vacio() ▷ O(1)
8: s.popularidad = 0 ▷ O(1)
9: s.huboConstruccion = false ▷ O(1)
10: s.mapasRecientesDeOtrosSC $\leftarrow$ vacio() ▷ O(1)

Complejidad: $O(1)$

---

iagregarCasa(**in/out** s : `SimCity`, **in** c : `casilla`)

1: definirRapido(s.casas,c,1) ▷ O(1)
2: s.huboConstruccion $\leftarrow$ true ▷ O(1)

Complejidad: $O(1)$

---

iagregarComercio(**in/out** s : `SimCity`,**in** c : `casilla`)

1: agregarRapido(s.comerciosRecientes,c) ▷ O(1)
2: s.huboConstruccion $\leftarrow$ true ▷ O(1)

Complejidad: $O(1)$

iavanzarTurno(**in/out** s : `SimCity`)

```
1: s.turno++                                                                          ▷ O(1)
2: if #s.comerciosRecientes != 0  then                                                ▷ O(1)
3:     itAConj ← crearIt(s.comerciosRecientes)                                        ▷ O(1)
4:     while haySiguiente(itAConj) do                                  ▷ O(#s.comerciosRecientes)
5:         nat max ← idarNivelAComercio(s,siguiente(itAConj))
6:         definirRapido(s.comercios,siguiente(itAConj),max)                          ▷ O(1)
7:         avanzar(itAConj)                                                           ▷ O(1)
8:         eliminarAnterior(itAConj)                                                  ▷ O(1)
9:     end while
10: end if
11: s.casas ← iunirCasasPriorizandoMaximoNivel(s)          ▷ O(iUnirCasasPriorizandoMaximoNivel(s))
12: s.comercios ← iunirComerciosPriorizandoMaximoNivel(s)  ▷ O(iUnirComerciosPriorizandoMaximoNivel(s))
13: itComRs ← crearIt(s.comerciosDeOtrosSc)                                           ▷ O(1)
14: while haySiguiente(itComRs) do                                  ▷ O(#s.comerciosDeOtrosSc)
15:     avanzar(itComRs)                                                              ▷ O(1)
16:     eliminarAnterior(itComRS)                                                     ▷ O(1)
17: end while
18: itTupl ← crearIt(s.casasYComerciosRecientesDeOtrosSC)                             ▷ O(1)
19: while haySiguiente(itTupl) do                  ▷ O(#s.casasYComerciosRecientesDeOtrosSC)
20:     itComR ← crearIt(siguiente(itTupl).first)                                     ▷ O(1)
21:     while haySiguiente(itComR) && !definido?(s.casas,siguiente(itComR)) do        ▷ O(#s.casas))
22:         definirRapido(s.comercios,siguiente(itComR),idarNivelAComercio(s,itComR))  ▷ O(idarNivelAComercio)
23:         avanzar(itComR)                                                           ▷ O(1)
24:     end while
25:     avanzar(itTupl)                                                               ▷ O(1)
26:     eliminarAnterior(itTupl)
27: end while
28: itADicc ← crearIt(s.casas)                                                        ▷ O(1)
29: while haySiguiente(itADicc) do                                          ▷ O(#s.casas)
30:     siguienteSignificado(itADicc)++                                               ▷ O(1)
31:     avanzar(itADicc)                                                              ▷ O(1)
32: end while
33: itADicc2 ← crearIt(s.comercios)                                                   ▷ O(1)
34: while haySiguiente(itADicc2) do                                         ▷ O(#s.comercios)
35:     siguienteSignificado(itADicc2)++                                              ▷ O(1)
36:     avanzar(itADicc2)                                                             ▷ O(1)
37: end while
38: itAConjMap ← crearIt(s.mapasRecientesDeOtrosSC)                                   ▷ O(1)
39: while haySiguiente(itAConjMap) do                       ▷ O(#s.mapasRecientesDeOtrosSC)
40:     iunirMapa(s1.mapa,siguiente(itAConjMap))
41:     avanzar(itAConjMap)                                                           ▷ O(1)
42:     eliminarAnterior(itAConjMap)                                                  ▷ O(1)
43: end while
44: s.huboConstruccion ← false                                                        ▷ O(1)
```

Complejidad: $O(\#\text{s.comerciosRecientes}) * idarNivelAComercio() + O(\text{iunirCasasPriorizandoMaximoNivel()}) + O(\text{iUnirComerciosPriozandoMaximoNivel()}) + O(\#\text{s.comercioDeOtrosSC}) + O(\#\text{s.casasYComerciosRecientesDeOtrosSC}) * O(\#\text{siguiente(itTupl).first} * \#\text{s.casas} * \text{idarNivelAComercio}) + O(\#\text{s.casas}) + O(\#\text{s.comercios}) + O(\#\text{s.mapasRecientesDeOtrosSC} * \text{uniMapa()})$

iunir(**in/out** $s1$: `SimCity`,**in** $s2$: `SimCity`)

```
1: if s1.turno < s2.turno then                                                          ▷ O(1)
2:     s1.turno ← s2.turno                                                              ▷ O(1)
3: end if
4: agregarRapido(s1.ComerciosDeOtrosSC,s2.comercios)                                    ▷ O(1)
5: s1.popularidad ← s1.popularidad + s2.popularidad + 1                                 ▷ O(1)
6: s1.comerciosDeOtrosSC ← iunion(s1.comerciosDeOtrosSC,s2.comerciosDeOtrosSC)          ▷ O(1)
7: s1.casasYComerciosRecientesDeOtrosSC ←
8:                 agregarRapido(iUnion(s1.casasYComerciosRecientesDeOtrosSC,
9:                 s2.CasasYComerciosRecientesDeOtrosSC),(s2.comerciosRecientes,s2.casas))   ▷ O()
10: iunionDeOtrosMapas(s1.mapasRecientesDeOtroSC,s2.mapasRecientesDeOtroSC)              ▷ O()
11: agregarRapido(s1.mapasRecientesDeOtroSC,s2.mapa)                                    ▷ O(1)
12: s1.huboConstruccion ← s1.huboConstruccion || s2.huboConstruccion                     ▷ O(1)
```

Complejidad: $O(copy(s2))$

---

iconstrucciones(**in** s : `SimCity`) → res : conj(Casilla)

```
1: return iunion(icasas(s),icomercios(s))                                               ▷ O()
```

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

---

ihayConstruccion(**in** s : `SimCity`,**in** c : `Casilla`) → res : bool

```
1: return pertenece?(iconstrucciones(s),c)                                  ▷ O(#construciones(s))
```

Complejidad: $O(\#construcciones(s))$

---

isePuedeConstruir(**in** s : `SimCity`,**in** c : `Casilla`) → res : bool

```
1: return !ihayRio?(imapa(s),c) && !ihayConstruccion(s,c))                              ▷ O()
```

Complejidad: $O(hayRio(mapa(s), c) + hayConstrucciones(s, c))$

---

iesCasillaDeMaximoNivel(**in** s : `SimCity`,**in** c : `Casilla`) → res : bool

```
1: return inivel(s,c) == maximoNivel(s,construcciones(s))
```

Complejidad: $O(inivel() + maximoNivel() + construcciones())$

---

iunirCasasPriorizandoNivelMaximo(**in** s : `SimCity`) → res : diccLineal(Casilla,nat)

```
1: res ← s1.casas                                                   ▷ O(#claves(s.casas))
2: itCasDeOtroSC ← crearIt(s1.CasasyComerciosRecientesDeOtroSC)                         ▷ O(1)
3: while haySiguiente(itCasDeOtroSC) do          ▷ O(# s.casasYComerciosRecientesDeOtrosSC)
4:     res ← iUnion(iunirCasas(res,siguiente(itCasDeOtroSC).second),
5:                 iunirCasas(siguiente(itCasDeOtroSC).second,res))                     ▷ O(−)
6:     avanzar(itCasDeOtroSC)                                                           ▷ O(1)
7: end while
8: return res                                                                           ▷ O(1)
```

Complejidad: $O((2*\#\text{s.casasYComerciosRecientesDeOtroSC}*\#\text{s.casas}^2) + (3 * \#s.CasasYComerciosRecientesDeOtroSC^2 * \#s.casas))$

---

iunirComerciosPriorizandoNivelMaximo(**in** s : `SimCity`) → res : diccLineal(Casilla,nat)

```
1: res ← s1.comercios                                                    ▷ O(copy(s1.comercios))
2: itComDeOtroSC ← crearIt(s1.ComerciosDeOtroSC)                         ▷ O(1)
3: while haySiguiente(itComDeOtroSC) do                                  ▷ O(#s1.comerciosDeOtrosSC)
4:     s1.comercios ← iUnion(iunirComercios(s1.comercios,siguiente(itComDeOtroSC)),
5:                           iunirComercios(siguiente(itComDeOtroSC),s1.comercios))   ▷ O()
6:     avanzar(itComDeOtroSC)
7: end while
8: return res                                                            ▷ O(1)
```

Complejidad: $O((2 * \#s.comerciosDeOtroSC * \#s.comercios^2) + (3 * \#s.comerciosDeOtroSC^2 * \#s.comercios))$

---

icasasADistancia3Manhattan(**in** cc : `conj(Casilla)`,**in** c : `Casilla`) → res : conj(Casilla)

```
1: res ← vacio()                                                         ▷ O(1)
2: if esVacio?(cc) then                                                  ▷ O(1)
3:     return res                                                        ▷ O(1)
4: else
5:     itConj ← crearIt(cc)                                              ▷ O(1)
6:     while haySiguiente?(itConj) do                                    ▷ O(#(cc))
7:         if idistanciaManhattan(siguiente(itConj),c) <= 3 then         ▷ O(1)
8:             agregarRapido(res,siguiente(itConj))                      ▷ O(1)
9:         end if
10:        avanzar(itConj)                                               ▷ O(1)
11:    end while
12: end if
13: return res                                                           ▷ O(1)
```

Complejidad: $O(\#cc)$

---

imaximoNivel(**in** s : `SimCity`,**in** c : `conj(Casilla)`) → res : nat

```
1: if # c != 0 then
2:     itConj ← crearIt(c)                                               ▷ O(1)
3:     maximo ← inivel(s,siguiente(itConj))
4:     avanzar(itConj)                                                   ▷ O(1)
5:     while haySiguiente(itConj) do                                     ▷ O(#(c))
6:         if maximo < inivel(s,siguiente(itConj)) then                  ▷ O(inivel())
7:             maximo ← inivel(s,siguiente(itConj))                      ▷ O(inivel)
8:         end if
9:     end while
10: else
11:    return 1                                                          ▷ O(1)
12: end if
13: return maximo                                                        ▷ O(1)
```

Complejidad: $O(\#c * inivel())$

---

iunirCasas(**in/out** $c1$: `diccLineal(Casilla,nat)`,**in** $c2$: `diccLineal(Casilla,nat)`)

```
1: itCon ← crearIt(claves(c1))                                                          ▷ O(1)
2: while haySiguiente(itConj) do                                                        ▷ O(#claves(c1))
3:     if  !pertenece?(claves(c2),(siguiente(itCon)) || (pertenece?(claves(c2),(siguiente(itCon))
4: && significado(c1,siguiente(itCon)) ≥ significado(c2,siguiente(itCon))) then      ▷ O(#claves(c1)+#claves(c2))
5:         definir(c1,siguiente(itCon),significado(c2,siguiente(itCon)))             ▷ O(#claves(c1)+#claves(c2))
6:     else
7:         borrar(c1,siguiente(itCon))                                               ▷ O(#claves(c1))
8:     end if
9:     avanzar(itCon)                                                                ▷ O(1)
10: end while
```

Complejidad: $O(\#claves(c1) * (\#claves(c1)) + \#claves(c2)))$

---

iunirComercios(**in/out** $c1$: `diccLineal(Casilla,nat)`,**in** $c2$: `diccLineal(Casilla,nat)`)

```
1: itCon ← crearIt(claves(c1))                                                          ▷ O(1)
2: while haySiguiente(itConj) do                                                        ▷ O(#claves(c1))
3:     if  !pertenece?(claves(c2),(siguiente(itCon)) || (pertenece?(claves(c2),(siguiente(itCon)) &&
4: significado(c1,siguiente(itCon)) >= significado(c2,siguiente(itCon))) then       ▷ O(#claves(c1)+#claves(c2))
5:         definir(c1,siguiente(itCon),significado(c2,siguiente(itCon))              ▷ O(#claves(c1)+#claves(c2))
6:     else
7:         borrar(c1,siguiente(itCon))                                               ▷ O(#claves(c1))
8:     end if
9:     avanzar(itCon)                                                                ▷ O(1)
10: end while
```

Complejidad: $O(\#claves(c1) * (\#claves(c1) + \#claves(c2)))$

---

idarNivelAComercio(**in** s : `SimCity`,**in** c : `Casilla`) → res : nat

```
1: dicc(Casilla,nat) resD ← iunircomerciosPriorizandoMaximoNivel(s)
2: res ← 0                                                                              ▷ O(1)
3: itManh ← crearIt(s.casasYComerciosRecientesDeOtrosSC)                                ▷ O(1)
4: while haySiguiente?(itManh) do                                                       ▷ O(#s.casasYComerciosRcientesDeOtrosSC)
5:     if pertenece?(siguiente(itManh).first,c)) then                                   ▷ O(# claves(siguiente(itManh)))
6:         res ← max{res,maxNivelManhattan(c,siguiente(itManh).second)}                 ▷ O(#claves(siguiente(itManh).second))
7:     end if
8:     avanzar(itManh)                                                                  ▷ O(1)
9: end while
10: if definido?(resD,c) then                                                           ▷ O(#claves(resD))
11:     return max{res,significado(resD,c)}                                             ▷ O(1)
12: else
13:     return res                                                                      ▷ O(1)
```

Complejidad: $O(\#s.casasYComerciosRecientesDeOtrosSC^2)$

---

imaxNivelManhattam(**in** c : `casilla`,**in** d : `dicc(casilla,nat)`) → res : nat

```
1: res ← 0                                                                              ▷ O(1)
2: 3Manh ← icasasADistancia3Manhattan(claves(d),c)                                      ▷ O(#claves(d))
3: itCas ← crearIT(3manh)                                                               ▷ O(1)
4: while haySiguiente?(itCas) do                                                        ▷ O(#3manh)
5:     res ← max{res,significado(d,siguiente(itCas))}                                   ▷ O(1)
6:     Avanzar(itCas)                                                                   ▷ O(1)
7: end while
8: return res                                                                           ▷ O(1)
```

Complejidad: $O(\#claves(d))$ =0

---

idistanciaManhattan(**in** x : `Casilla`,**in** y : `Casilla`) $\rightarrow$ res : nat

1: **return** $|x.first - y.first| + |x.second - y.second|$ ▷ O(1)

Complejidad: $O(1)$

---

iUnion(**in/out** $c1$: `conjLineal(Casilla)`,**in** $c2$: `conjLineal(Casilla)`)

1: itc2 ← crearIt(c2) ▷ O(1)
2: **while** haySiguiente(itc2) **do** ▷ O(# c2)
3: AgregarRapido(c1,siguiente(itc2)) ▷ O(1)
4: avanzar(itc2) ▷ O(1)
5: **end while**

Complejidad: $O(\#c2)$

---

iUnionDeOtrosMapas(**in/out** $c1$: `conjLineal(mapa)`,**in** $c2$: `conjLineal(mapa)`)

1: itc2 ← crearIt(c2) ▷ O(1)
2: **while** haySiguiente(itc2) **do** ▷ O(# c2)
3: AgregarRapido(c1,siguiente(itc2)) ▷ O(1)
4: avanzar(itc2) ▷ O(1)
5: **end while**

Complejidad: $O(\#c2)$

---

3. Diccionario

Interfaz

CLAVES(**in** d: `dicc(k,s)`) $\rightarrow$ res : `conj(k)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} claves(s)$}
Complejidad: $O(\#claves(d)$
Descripción: Devuelve las claves del diccionario

UNION(**in/out** $c1$: `dicc(k,s)`,**in** $c2$: `dicc(k,s)`)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$}
Complejidad: $O(\#claves(c2))$
Descripción: Une dos diccionarios

Algoritmos

---

claves(**in** d: `dicc(k,s)`) $\rightarrow$ res :conj(k)

1: itDicc ← crearIt(d)
2: res ← vacio()
3: **while** haySiguiente(itDicc) **do**
4: agregarRapido(res,siguienteClave(itDicc))
5: **end while**
6: **return** res

Complejidad: $O(\#claves(d))$

Union(**in/out** $c1$: `diccLineal(k,s)`,**in** $c2$: `diccLineal(k,s)`)

```
1: itc2 ← crearIt(c2)                                                    ▷ O(1)
2: while haySiguiente(itc2) do                                           ▷ O(# claves(c2))
3:     definirRapido(c1,siguienteclave(itc2),siguientesignificado(itc2)) ▷ O(1)
4:     avanzar(itc2)                                                     ▷ O(1)
5: end while
```

Complejidad: $O(\#claves(c2))$