1. Especificación

TAD MAPA

igualdad observacional

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

géneros Mapa

exporta Mapa, generadores, observadores

usa NAT, CONJUNTO(α)

observadores básicos

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

Mapa

generadores

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

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

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

Fin TAD

TAD Pos es tupla(Nat, Nat)

TAD Nivel es Nat

TAD Construccion es Sting

TAD SimCity

igualdad observacional

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

géneros SimCity

exporta SimCity, generadores, observadores, otras operaciones

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

observadores básicos

mapa : SimCity $\longrightarrow$ Mapa

casas : SimCity $\longrightarrow$ dicc(Pos, Nivel)

comercios : SimCity $\longrightarrow$ dicc(Pos, Nivel)

popularidad : SimCity $\longrightarrow$ Nat

generadores

iniciar : Mapa $\longrightarrow$ SimCity

avanzarTurno : SimCity s $\times$ dicc(Pos $\times$ Construccion) cs $\longrightarrow$ SimCity

$$\left\{ \begin{array}{l} 1 \leq \#\text{claves(cs)} \wedge_{\text{L}} (\forall p\text{: Pos})(\text{p} \in \text{claves(cs)} \Rightarrow_{L} \text{p} \notin (\text{claves(casas(s))} \cup \text{claves(comercios(s))})) \\ \wedge\ \text{p} \notin (\text{horizontales(mapa(s))} \cup \text{verticales(mapa(s))})) \end{array} \right\}$$

unir : SimCity a $\times$ SimCity b $\longrightarrow$ SimCity

$$\left\{ \begin{array}{l} //\text{ríos no eliminan construcciones:} \\ (\forall\ \text{p} : \text{Pos})(\text{p} \in (\text{claves(casas(a))} \cup \text{claves(comercios(a))}) \Rightarrow \\ (\pi_1(p) \notin \text{horizontales(mapa(b))} \wedge \pi_2(p) \notin \text{verticales(mapa(b))})) \wedge \\ (\forall\ \text{p} : \text{Pos})(\text{p} \in (\text{claves(casas(b))} \cup \text{claves(comercios(b))}) \Rightarrow \\ (\pi_1(p) \notin \text{horizontales(mapa(a))} \wedge \pi_2(p) \notin \text{verticales(mapa(a))})) \\ \\ //\text{no se pisan construcciones de nivel máximo:} \\ \text{vacio?(constrDeNivel(a, maxNivel(a)) } \cap \text{ (casas(b) } \cup \text{ comercios(b))) } \wedge \\ \text{vacio?(constrDeNivel(b, maxNivel(b)) } \cap \text{ (casas(a) } \cup \text{ comercios(a))) } \end{array} \right\}$$

otras operaciones

turno : SimCity $\longrightarrow$ Nat

antigüedad : SimCity $\longrightarrow$ Nat

distManhattan3 : Pos $\times$ Pos $\longrightarrow$ Bool

esCasa? : Construccion $\longrightarrow$ Bool

esComercio? : Construccion $\longrightarrow$ Bool

SubirNiveles : dicc(Pos $\times$ Nivel) $\longrightarrow$ dicc(Pos, Nivel)

AgCasas : dicc(Pos $\times$ Construccion) dc $\times$ dicc(Pos $\times$ Nivel) cs $\longrightarrow$ dicc(Pos,Nivel) {(∀ p: Pos)(p ∈ claves(dc) $\Rightarrow_L$ p ∉ (claves(dca) ∪ claves(cs)))}

AgComercios : dicc(Pos $\times$ Construccion) dc $\times$ dicc(Pos $\times$ Nivel) dca $\times$ dicc(Pos $\times$ Nivel) cs $\longrightarrow$ dicc(Pos,Nivel) {(∀ p: Pos)(p ∈ claves(dc) $\Rightarrow_L$ p ∉ (claves(dca) ∪ claves(cs)))}

NivelNuevoCom : Pos p $\times$ dicc(Pos $\times$ Construccion) d $\longrightarrow$ Nat {p∉claves(d)}

unirConstrucciones : dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Nivel) $\longrightarrow$ dicc(Pos, Nivel)

actualizarNivelComercios : dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Nivel) $\longrightarrow$ dicc(Pos, Nivel)

maxNivel : SimCity $\longrightarrow$ Nat

maxAux : dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Nivel) $\times$ Nat $\longrightarrow$ Nat

constrDeNivel : SimCity × Nivel ⟶ Conj(Pos)
constrDeNAux : dicc(Pos × Nivel) × dicc(Pos × Nivel) × Nivel × Conj(Pos) ⟶ Conj(Pos)

axiomas $\forall s, s', a, b$: simcity, $\forall dc$: dicc(Pos, Construccion), $\forall dca, cs, dc1, dc2, dco$: dicc(Pos, Nivel), $\forall p, q$: Pos,
$\forall c$: Construccion, $\forall m$: Mapa, $\forall cRes$: Conj(Pos), $\forall maxRes, niv, n$: Nat

mapa(iniciar(m)) ≡ m
mapa(avanzarTurno(s, dc)) ≡ mapa(s)
mapa(unir(a, b)) ≡ crear(horizontales(mapa(a)) ∪ horizontales(mapa(b)), verticales(mapa(a)) ∪ verticales(mapa(b)))

casas(iniciar(m)) ≡ vacio()
casas(avanzarTurno(s, dc)) ≡ AgCasas(dc, SubirNiveles(casas(s)))
casas(unir(a, b)) ≡ unirConstrucciones(casas(a), casas(b))

comercios(iniciar(m)) ≡ vacio()
comercios(avanzarTurno(s, dc)) ≡ AgComercios(dc, AgCasas(dc, SubirNiveles(casas(s))), SubirNiveles(comercios(s)))
comercios(unir(a, b)) ≡ actualizarNivelComercios(unirConstrucciones(comercios(a), comercios(b)), unirConstrucciones(casas(a),casas(b)))

popularidad(iniciar(m)) ≡ 0
popularidad(avanzarTurno(s, dc)) ≡ popularidad(s)
popularidad(unir(a, b)) ≡ popularidad(a)+popularidad(b)+1

turno(iniciar(m)) ≡ 0
turno(avanzarTurno(s, dc)) ≡ turno(s)+1
turno(unir(a,b)) ≡ turno(a)+1

antigüedad(s) ≡ máx(maxAux(casas(s), vacio(), 0), maxAux(comercios(s), vacio(), 0))

distManhattan3(p, q) ≡ $|p_0\text{-}q_0|+|p_1\text{-}q_1| \leq 3$

esCasa?(c) ≡ c = "casa"
esComercio?(c) ≡ c = "comercio"

unirConstrucciones(dc1, dc2) ≡
if ∅?(claves(dc2)) **then**
  dc1
else
  if ¬ definido?(dameUno(claves(dc2)),dc1) **then**
    unirConstrucciones(definir(dameUno(claves(dc2)), obtener(dameUno(claves(dc2)),dc2),dc1), borrar(dameUno(claves(dc2)), dc2))
  else
    unirConstrucciones(dc1, borrar(dameUno(claves(dc2)), dc2))
  fi
fi

actualizarNivelComercios(dco, dca) ≡
if ∅?(claves(dca)) **then**
  dco
else
  definir(dameUno(claves(dco)), máx(nivelNuevoComercio(dameUno(claves(dco)),dca), obtener(dameUno(claves(dco)))), actualizarNivelComercios(borrar(dameUno(claves(dco)),dco),dca))
fi

```
AgCasas(dc, cs)  ≡ if ∅?(claves(dc)) then
                     cs
                   else
                     if esCasa?(obtener(dameUno(claves(dc))),dc) then
                       definir(dameUno(claves(dc)), 1,
                           AgCasas(borrar(dameUno(claves(dc)), dc),cs))
                     else
                       AgCasas(borrar(dameUno(claves(dc)), dc), cs)
                     fi
                   fi

AgComercios(dc, dca, cs)  ≡ if ∅?(claves(dc)) then
                              cs
                            else
                              if esComercio?(obtener(dameUno(claves(dc))),dc) then
                                definir((dameUno(claves(dc))),
                                  NivelNuevoComercio(dameUno(claves(dc)), dca),
                                  AgComercios(borrar(dameUno(claves(dc)), dc), dca, cs))
                              else
                                AgComercios(borrar(dameUno(claves(dc)), dc), dca, cs)
                              fi
                            fi

NivelNuevoComercio(p, dca)  ≡ if ∅?(claves(dca)) then
                                1
                              else
                                if distManhatan3(p, dameUno(claves(dca))) then
                                  max(obtener(dameUno(claves(dca)), dca),
                                      NivelNuevoComercio(p,
                                      borrar(dameUno(claves(dca)), dca))
                                else
                                  NivelNuevoComercio(p,
                                      borrar(dameUno(claves(dca)), dca))
                                fi
                              fi

SubirNiveles(dc)  ≡ if ∅?(claves(dc)) then
                      vacio()
                    else
                      definir(unaClave(dc),
                              obtener(unaClave(dc), dc) + 1,
                              SubirNiveles(borrar(unaClave(dc)), dc))
                    fi

maxNivel(s)  ≡ maxAux(casas(s), comercios(s), 0)

maxAux(dca, dc, maxRes)  ≡ if ∅?(claves(dca)) ∧ ∅?(claves(dc)) then
                             maxRes
                           else
                             if ¬ ∅?(claves(dca)) then
                               if obtener(dameUno(claves(dca)), dca) > maxRes then
                                 maxAux(borrar(dameUno(claves(dca)), dca), dc,
                                         obtener(dameUno(claves(dca)), dca))
                               else
                                 maxAux(borrar(dameUno(claves(dca)), dca), dc, maxRes)
                               fi
                             else
                               if obtener(dameUno(claves(dc)), dc) > maxRes then
                                 maxAux(dca, borrar(dameUno(claves(dc)), dc),
                                         obtener(dameUno(claves(dc)), dcc))
                               else
                                 maxAux(dca, borrar(dameUno(claves(dc)), dc), maxRes)
                               fi
                             fi
                           fi
```

```
constrDeNivel(s, niv)          ≡ constrDeNivelAux(casas(s), comercios(s), niv, ∅)

constrDeNivelAux(dca, dc, n, cRes) ≡ if ∅?(claves(dca)) ∧ ∅?(claves(dc)) then
                                        cRes
                                     else
                                        if ¬ ∅?(claves(dca)) then
                                           if obtener(dameUno(claves(dca)), dca) = n then
                                              constrDeNivelAux(borrar(dameUno(claves(dca)),
                                              dca),                dc, n, Ag(dameUno(claves(dca)),
                                              cRes))
                                           else
                                              constrDeNivelAux(borrar(dameUno(claves(dca)),
                                              dca),                  dc, n, cRes)
                                           fi
                                        else
                                           if obtener(dameUno(claves(dc)), dc) = n then
                                              constrDeNivelAux(dca,borrar(dameUno(claves(dc)),
                                                           dc), n, Ag(dameUno(claves(dc)), cRes))
                                           else
                                              constrDeNivelAux(dca,borrar(dameUno(claves(dc)),
                                                             dc), n, cRes)
                                           fi
                                        fi
                                     fi
```

Fin TAD

Observación 1: Tomamos que el turno es distinto a la antigüedad. Si bien, la antigüedad se define como la cantidad de turnos que pasaron, ésta puede cambiar al unir dos simcity's, por lo que elegimos que más allá de esto, no cambien los turnos del simcity actual.

Observación 2: Decidimos que al unir dos SimCity's, si se superponen construcciones, queden las de la primera Simcity.

TAD Nombre es String

TAD SERVIDOR

igualdad observacional

$$(\forall s, s' : \text{Servidor}) \left(s =_{\text{obs}} s' \iff \begin{pmatrix} \text{juegos(s)} =_{\text{obs}} \text{juegos(s')} \wedge_{\text{L}} \\ (\forall\ \text{j} : \text{juego})(\text{j} \in \text{juegos(s)} \ \Rightarrow_{L} \\ \text{simcity(s, j)} =_{\text{obs}} \text{simcity(s', j)}) \end{pmatrix} \right)$$

géneros Servidor

exporta Servidor, generadores, observadores, otras operaciones

usa SIMCITY, MAPA, NOMBRE, CONJUNTO(α), POS, CONSTRUCCION, DICCIONARIO(α)

observadores básicos

juegos : Servidor $\longrightarrow$ conj(nombre)

simcity : Servidor sv $\times$ nombre n $\longrightarrow$ SimCity $\{\text{n} \in \text{juegos(sv)}\}$

generadores

crearServidor : $\longrightarrow$ Servidor

registrar : Servidor sv $\times$ nombre n $\times$ Mapa $\longrightarrow$ Servidor $\{\text{n} \notin \text{juegos(sv)}\}$

avanzarTurnoSimcity : Servidor sv $\times$ nombre n $\times$ dicc(Pos $\times$ Construccion) dc $\longrightarrow$ Servidor

$$\begin{cases} \text{n} \in \text{juegos(sv)} \wedge 1 \leq \#\text{claves(dc)} \wedge_{\text{L}} \\ (\forall p\text{: Pos})(\text{p} \in \text{claves(dc)} \Rightarrow_{L} \\ \text{p} \notin (\text{claves(casas(sv, n))} \cup \text{claves(comercios(sv, n))}) \wedge \\ \text{p} \notin (\text{horizontales(mapa(sv, n))} \cup \text{verticales(mapa(sv, n))}) \end{cases}$$

unirSimcitys : Servidor sv $\times$ nombre $n1$ $\times$ nombre $n2$ $\longrightarrow$ Servidor

$$\begin{cases} \{\text{n1,n2}\} \subseteq \text{juegos(sv)} \wedge \\ //\text{ríos no eliminan construcciones:} \\ (\forall\ \text{p} : \text{Pos})(\text{p} \in (\text{claves(casas(sv, n1))} \cup \text{claves(comercios(sv, n1))}) \ \Rightarrow \\ (\pi_1(p) \notin \text{horizontales(mapa(sv, n2))} \wedge \pi_2(p) \notin \text{verticales(mapa(sv, n2))})) \wedge \\ (\forall\ \text{p} : \text{Pos})(\text{p} \in (\text{claves(casas(sv, n2))} \cup \text{claves(comercios(simcity(sv, n2))))}) \ \Rightarrow \\ (\pi_1(p) \notin \text{horizontales(mapa(sv, n1))} \wedge \pi_2(p) \notin \text{verticales(mapa(sv, n1))})) \\ \\ //\text{no se pisan construcciones de nivel máximo:} \\ \text{vacio?(constrDeNivel(simcity(sv, n1), maxNivel(simcity(sv, n1))) } \cap \text{ (casas(sv, n2) } \cup \\ \text{comercios(sv, n2))) } \wedge \\ \text{vacio?(constrDeNivel(simcity(sv, n2), maxNivel(simcity(sv, n2))) } \cap \text{ (casas(sv, n1) } \cup \\ \text{comercios(sv, n1)))} \end{cases}$$

otras operaciones

mapaSimcity : Servidor sv $\times$ Nombre n $\longrightarrow$ Mapa $\{\text{n} \in \text{juegos(sv)}\}$

casasSimcity : Servidor sv $\times$ Nombre n $\longrightarrow$ dicc(Pos, Nivel) $\{\text{n} \in \text{juegos(sv)}\}$

comerciosSimcity : Servidor sv $\times$ Nombre n $\longrightarrow$ dicc(Pos, Nivel) $\{\text{n} \in \text{juegos(sv)}\}$

popularidadSimcity : Servidor sv $\times$ Nombre n $\longrightarrow$ Nat $\{\text{n} \in \text{juegos(sv)}\}$

antigüedadSimcity : Servidor sv $\times$ Nombre n $\longrightarrow$ Nat $\{\text{n} \in \text{juegos(sv)}\}$

turnoSimcity : Servidor sv $\times$ Nombre n $\longrightarrow$ Nat $\{\text{n} \in \text{juegos(sv)}\}$

axiomas $\forall sv$: Servidor, $\forall n, n', n1, n2$: Nombre, $\forall m$: Mapa, $\forall dc$: dicc(Pos, Construccion)

juegos(crearServidor()) $\equiv$ $\emptyset$

juegos(registrar(sv, n, m)) $\equiv$ Ag(n, juegos(sv))

juegos(avanzarTurnoSimcity(sv, n, dc)) $\equiv$ juegos(sv)

juegos(unirSimcitys(sv, n1, n2)) $\equiv$ juegos(sv)

simcity(registrar(sv, n, m), n') ≡ **if** n $=_{obs}$ n' **then** iniciar(m) **else** simcity(sv, n') **fi**
simcity(avanzarTurnoSimcity(sv, n, dc), n') ≡ **if** n $=_{obs}$ n' **then**
avanzarTurno(simcity(sv, n), dc)
else
simcity(sv, n')
fi
simcity(unirSimcitys(sv, n1, n2), n') ≡ **if** n1 $=_{obs}$ n' **then**
unir(simcity(sv, n1), simcity(sv, n2))
else
simcity(sv, n')
fi

mapaSimcity(sv, n) ≡ mapa(simcity(sv, n))

casasSimcity(sv, n) ≡ casas(simcity(sv, n))

comerciosSimcity(sv, n) ≡ comercios(simcity(sv, n))

popularidadSimcity(s,n) ≡ popularidad(simcity(sv, n))

antigüedadSimcity(s,n) ≡ antigüedad(simcity(sv, n))

turnoSimcity(sv, n) ≡ turno(simcity(sv, n))

Fin TAD

2. Módulos de referencia

2.1. Módulo Mapa

Interfaz

se explica con: MAPA

géneros: `mapa`

Trabajo Práctico Nro 1Operaciones básicas de mapa

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

HORIZONTALES(**in** m: `mapa`) $\rightarrow$ res : `conj(Nat)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} horizontales(m)\}$
Complejidad: $O(copy(hs))$
Descripción: Devuelve los rios horizontales del mapa pasado como parámetro.

VERTICALES(**in** m: `mapa`) $\rightarrow$ res : `conj(Nat)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} verticales(m)\}$
Complejidad: $O(copy(vs))$
Descripción: Devuelve los rios verticales del mapa pasado como parámetro.

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) $\equiv$ m' : *estr*/
horizontales(m) =obs m'.horizontales $\wedge$ verticales(m) =obs m'.verticales

Algoritmos

---

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

1: $estr.horizontales \leftarrow hs$
2: $estr.verticales \leftarrow vs$
3: **return** estr

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

---

horizontales(**in** m : `mapa`) $\rightarrow$ res : conj(Nat)

1: **return** $\leftarrow m.horizontales$

Complejidad: $O(copy(m.horizontales))$

---

verticales(**in** m : `mapa`) $\rightarrow$ res : conj(Nat)

1: **return** $\leftarrow m.verticales$

Complejidad: $O(copy(m.verticales))$

2.2. Módulo SimCity

Interfaz

se explica con: SimCity

géneros: `simcity`

Operaciones básicas de sc

INICIAR(**in** m : `mapa`) → res : `simcity`
Pre ≡ {true}
Post ≡ {res=$_{obs}$ iniciar(m)}
Complejidad: $O(copy(m))$
Descripción: Inicia un simcity con el mapa pasado como parámetro.

AVANZARTURNO(**in/out** s : `simcity`, **in** cs : `dicc(Pos, Construccion)`)
Pre ≡ {$s_0 = s \wedge$ ($\forall p$: Pos)(def?(p, cs) $\Rightarrow_L$ $p_0 \notin$ horizontales(mapa(s)) $\wedge$ $p_1 \notin$ verticales(mapa(s)) $\wedge$
$p \notin$ (claves(casas(s)) $\cup$ claves(comercios(s)))}
Post ≡ {s =$_{\text{obs}}$ avanzarTurno(s_0, cs)}
Complejidad: $O(\#claves(cs))$
Descripción: Modifica el simcity pasado como parámetro, agregando las nuevas construcciones con sus respectivos niveles y actualizando las que ya están.

UNIR(**in/out** a : `simcity`, **in** b : `simcity`)
Pre ≡ {a_0 = a $\wedge$
//ríos no eliminan construcciones:
($\forall$ p : Pos)(p $\in$ (claves(casas(a)) $\cup$ claves(comercios(a))) $\Rightarrow$
$p_0 \notin$ horizontales(mapa(b)) $\wedge$ $p_1 \notin$ verticales(mapa(b)))) $\wedge$
($\forall$ p : Pos)(p $\in$ (claves(casas(b)) $\cup$ claves(comercios(b))) $\Rightarrow$
($p_0 \notin$ horizontales(mapa(a)) $\wedge$ $p_1 \notin$ verticales(mapa(a))))

//no se pisan construcciones de nivel máximo:
vacio?(constrDeNivel(a, maxNivel(a)) $\cap$ (casas(b) $\cup$ comercios(b))) $\wedge$
vacio?(constrDeNivel(b, maxNivel(b)) $\cap$ (casas(a) $\cup$ comercios(a)))}
Post ≡ {a =$_{\text{obs}}$ unir(a_0, b)}
Complejidad: $O(1)$
Descripción: Modifica el primer simcity pasado como parámetro, de modo que en éste queden como propios los rios y las construcciones del segundo simcity pasado como parámetro.

MAPA(**in** s : `simcity`) → res : `mapa`
Pre ≡ {true}
Post ≡ {alias(res=$_{obs}$ s.mapa)}
Complejidad: $O(copy(s.mapa) + Longitud(s.unidoConSimcitys)$ x
$\sum_{i=0}^{Longitud(s.unidoConSimcitys)}((((s.unidoConSimcity[i])_0).mapa).verticales\ +$
$(((s.unidoConSimcity[i])_0).mapa).horizontales)$) [
Descripcion: Devuelve el mapa del simcity pasado como parámetro.]

CASAS(**in** s : `simcity`) → res : `dicc(Pos, Nivel)`
Pre ≡ {true}
Post ≡ {res=$_{obs}$ s.casas}
Complejidad: $O(Longitud(s.casas) + Longitud(s.unidoConSimcitys)$ x
$\sum_{i=0}^{Longitud(s.unidoConSimcitys)} \#claves(casas((s.unidoConSimcitys[i])_0)))]$ [
Descripcion: Devuelve las casas del simcity pasado como parámetro, con sus respectivos turnos de aparición en el simcity.]

COMERCIOS(**in** s : `simcity`) → res : `dicc(Pos, Nivel)`
Pre ≡ {true}
Post ≡ {res=$_{obs}$ s.comercios}
Complejidad: $O(Longitud(s.comercios) + Longitud(s.unidoConSimcitys)$ x
$\sum_{i=0}^{Longitud(s.unidoConSimcitys)} \#claves(comercios((s.unidoConSimcitys[i])_0))))]$ [
Descripcion: Devuelve los comercios del simcity pasado como parámetro, con sus respectivos niveles.]

POPULARIDAD(**in** s: `simcity`) $\rightarrow$ *res* : `Nat`
Pre $\equiv$ {true}
Post $\equiv$ {res=$_{obs}$ s.popularidad}
Complejidad: $O(1)$
Descripción: Devuelve la popularidad del simcity pasado como parámetro.

Donde Nombre es el nombre mas largo de los SimCitys.

Representación

Representación de simcity

`simcity` **se representa con** `estr`

donde `estr` es `tupla`(*mapa*: `mapa`,
casas: `listaEnlazada(tupla(Pos, TurnoAparicion))`,
comercios: `listaEnlazada(tupla(Pos, TurnoAparicion))`,
unidoConSimcitys: `listaEnlazada(tupla(puntero(simcity), turnoDeUnion))`,
popularidad: `Nat`, *antigüedad*: `Nat`, *turno*: `Nat`)

Donde Pos es tupla(Nat, Nat).

Donde TurnoAparicion es Nat, y representa el turno en el que fue creada la construccion en el simcity.

Donde turnoDeUnion es Nat, y representa el turno que tenía el SimCity al que se le aplicó la unión, cuando ésta se efectuó con el SimCity indicado por el puntero

Observación: Tomamos que el turno es distinto a la antigüedad. Si bien, la antigüedad se define como la cantidad de turnos que pasaron, ésta puede cambiar al unir dos simcity's, por lo que elegimos que más allá de esto, no cambien los turnos del simcity actual.

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\Longleftrightarrow$

▷ Las construcciones no están ubicadas sobre rios.

($\forall$ c : tupla(Pos, TurnoAparicion))(esta?(c, e.casas & e.comercios) $\Rightarrow$
($\pi_1(\pi_1(c)) \notin$ (e.mapa).horizontales $\wedge$ $\pi_2(\pi_1(c)) \notin$ (e.mapa).verticales))

$\wedge$

▷ No hay una casa y un comercio en la misma posición. Y no hay dos casas y dos comercios con distintos niveles en la misma posición.

($\forall$ c : tupla(Pos, TurnoAparicion))(esta?(c, e.casas) $\Rightarrow$
$\neg(\exists t$:TurnoAparicion)(esta?($\langle\pi_1(c), t\rangle$, e.comercios) $\vee$ (t $\neq \pi_2(c) \wedge$ está?($\langle\pi_1(c), t\rangle$, e.casas))
($\forall$ c : tupla(Pos, TurnoAparicion))(esta?(c, e.comercios) $\Rightarrow$
$\neg(\exists t$:TurnoAparicion)(esta?($\langle\pi_1(c), t\rangle$, e.casas) $\vee$ (t $\neq \pi_2(c) \wedge$ está?($\langle\pi_1(c), t\rangle$, e.comercios))

$\wedge$

▷ El nivel de todas las casas y comercios está entre 1 y turnos.

($\forall$ c : tupla(Pos, TurnoAparicion))((está?(c, e.casas) $\vee$ está?(c, e.comercios)) $\Rightarrow$ $1 \leq \pi_2(c) \leq$ e.turnos)

$\wedge$

▷ Hay al menos una construcción en cada turno.
($\forall$ t : nat)($1 \leq t \leq$ e.turnos) $\Rightarrow$ ($\exists$ p : Pos)(está?($\langle p, t\rangle$, e.casas) $\vee$ está?($\langle p, t\rangle$, e.comercios))

$\wedge$

▷ Las casas y comercios se encuentran ordenadas por turnoDeAparicion
ordenado(e.casas) $\wedge$ ordenado(e.comercios)

$\wedge$

▷ Las construcciones de las uniones no se solapan con rios.
($\forall$ s1,s2 : tupla(puntero(simcity), turnoDeUnion))
(está?(s1, e.unidoConSimcitys) $\wedge$ está?(s2, e.unidoConSimcitys) $\wedge \neg$(s1 $=_{\text{obs}}$ s2) $\Rightarrow$
($\forall$ c : tupla(Pos, TurnoAparicion))(esta?(c, s1 $\rightarrow$ casas & s1 $\rightarrow$ comercios) $\Rightarrow$
($\pi_1(c) \notin$ (s2 $\rightarrow$ mapa).verticales $\wedge$ $\pi_2(c) \notin$ (s2 $\rightarrow$ mapa).horizontales)

$\wedge$

▷ No se generan ciclos.
noEsUnion(e.UnidoConSimcitys, e)) $\wedge$
($\forall$ s, s' : tupla(puntero(simcity), turnoDeUnion))
(está?(s,e.UnidoConSimcitys) $\wedge$ está?(s',e.UnidoConSimcitys) $\wedge \neg$(s $=_{\text{obs}}$ s')
$\Rightarrow$ noEsUnion(π_1(s) $\rightarrow$ UnidoConSimcitys, s'))

$\wedge$

▷ Con respecto a antigüedad
e.turnos $\leq$ e.antigüedad $\wedge$
($\forall$ s : tupla(puntero(simcity), turnoDeUnion))(está?(s, e.UnidoConSimcitys) $\Rightarrow$
(π_1(s) $\rightarrow$ antigüedad) + e.turnos - π_2(s) $\leq$ e.antigüedad)
$\wedge$ (e.turnos = e.antigüedad $\vee$
($\exists$ s : tupla(puntero(simcity), turnoDeUnion))(está?(s, e.UnidoConSimcitys) $\wedge$
(π_1(s) $\rightarrow$ antigüedad) + e.turnos - π_2(s) = e.antigüedad))

$\wedge$

▷ Con respecto a popularidad
e.popularidad = $1 + \sum_{i=0}^{long(e.UnidoConSimcitys)} \pi_1$(valorEnIndice(e.UnidoConSimcitys,i)).popularidad

ordenado : secu(tupla(Pos $\times$ turnoDeAparicion)) $\longrightarrow$ Bool
ordenado(s) $\equiv$ long(s) $\leq 1 \vee_{\text{L}}$ (π_2(prim(s)) $\leq \pi_2$(prim(fin(s))) $\wedge$ ordenado(fin(s)))

valorEnIndice : secu(α) s $\times$ nat n $\longrightarrow$ α {n <long(s)}
valorEnIndice(s,n) $\equiv$ **if** n = 0 **then** prim(s) **else** valorEnIndice(fin(s), n-1) **fi**

noEsUnion : secu(puntero(simcity) $\times$ turtnoDeUnion) s $\times$ simcity sc $\longrightarrow$ bool
noEsUnion(s, sc) $\equiv$ vacio?(s) $\vee_{\text{L}}$ ($\neg$(*(π_1(prim(s)) $=_{\text{obs}}$ sc) $\wedge$ noEsUnion(π_1(prim(s) $\rightarrow$ UnidoConSimcitys, sc)
$\wedge$ noEsUnion(fin(s),sc))

Abs : estr s $\longrightarrow$ simcity {Rep(s)}
Abs(s) $\equiv$ $t : estr /$ mapa(s) $=_{obs}$ t.mapa
$\wedge$ casas(s) $=_{obs}$ UnirConstrucciones(hacerDicc(t.casas, t.turno),
UnirTodos(t.unidoConSimcitys))
$\wedge$ comercios(s) $=_{obs}$ ActualizarNivelComercios(UnirConstrucciones(hacerDicc(t.comercios, t.turno),
UnirTodosComercios(t.unidoConSimcitys)), casas(s))
$\wedge$ popularidad(s) $=_{obs}$ t.popularidad
$\wedge$ antigüedad(s) $=_{obs}$ t.antigüedad
$\wedge$ turno(s) $=_{obs}$ t.turno

unirTodosCasas : secu(tupla(puntero(simcity)) $\times$ turnoDeAparicion) $\longrightarrow$ dicc(Pos,Nivel)
unirTodosCasas(s) $\equiv$ **if** vacío?(s) **then**
vacío()
else
unirConstrucciones(casas(Abs(π_1(prim(s)))), unirTodosCasas(fin(s)))
fi

unirTodosComercios : secu(tupla(puntero(simcity)) $\times$ turnoDeAparicion) $\longrightarrow$ dicc(Pos,Nivel)
unirTodosComercios(s) $\equiv$ **if** vacío?(s) **then**
vacío()
else
unirConstrucciones(comercios(Abs(π_1(prim(s)))), unirTodosComercios(fin(s)))
fi

hacerDicc : secu(tupla(Pos $\times$ turnoDeAparicion)) $\times$ nat $\longrightarrow$ dicc(Pos, Nivel)
hacerDicc(s,t) $\equiv$ **if** vacío(s) **then** vacío() **else** definir(π_1(prim(s)), t -π_2(prim(s)), hacerDicc(fin(s), t) **fi**

Algoritmos

---

iniciar(**in** m : `mapa`) $\rightarrow$ res : simcity

1: $res.mapa \leftarrow m$
2: $res.casas \leftarrow$ Vacía()
3: $res.comercios \leftarrow$ Vacía()
4: $res.unidoConSimcitys \leftarrow$ Vacía()
5: $res.popularidad \leftarrow 0$
6: $res.antigüedad \leftarrow 0$
7: $res.turnos \leftarrow 0$
8: **return** res

Complejidad: $O(copy(m))$

---

avanzarTurno(**in**/**out** s : `simcity`, **in** cs : `dicc(Pos, Construccion)`)

1: $it \leftarrow CrearIt(cs)$
2: **while** haySiguiente(it) **do**
3: **if** it.siguienteSignificado == casa **then**
4: $AgregarAtras(s.casas, (it.siguienteClave, s.turno))$
5: **else**
6: $AgregarAtras(s.comercios, (it.siguienteClave, s.turno))$
7: **end if**
8: $avanzar(it)$
9: **end while**
10: $s.turnos \leftarrow s.turnos + 1$
11: $s.antigüedad \leftarrow s.antigüedad + 1$

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

Observacion: En línea 4 se cumple que agregar una casa es O(1). Análogamente, en línea 6 se cumple que agregar un comercio es O(1)

---

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

1: $AgregarAtras(a.unidoConSimcitys, (\&b, a.turno))$
2: $a.popularidad \leftarrow a.popularidad + 1$
3: $a.antigüedad \leftarrow max(a.antigüedad, b.antigüedad)$
4: $a.turno \leftarrow a.turno + 1$

Complejidad: $O(1)$

---

mapa(**in** s : `simcity`) $\rightarrow$ res : mapa

1: $res \leftarrow s.mapa$
2: $it \leftarrow CrearIt(s.unidoConSimcitys)$
3: **while** haySiguiente(it) **do**
4: $itv \leftarrow CrearIt(mapa(siguiente(it)).verticales)$
5: **while** haySiguiente(itv) **do**
6: $Agregar(res, siguiente(itv))$
7: $Avanzar(itv)$
8: **end while**
9: $ith \leftarrow CrearIt(mapa(siguiente(it)).horizontales)$
10: **while** haySiguiente(ith) **do**
11: $Agregar(res, siguiente(ith))$
12: $Avanzar(ith)$
13: **end while**
14: **end while**
15: **return** res

16: Complejidad: O(copy(s.mapa) + Longitud(s.unidoConSimcitys) x
17: $\sum_{i=0}^{Longitud(s.unidoConSimcitys)}$((((s.unidoConSimcity[i])$_0$).mapa).verticales +
18: (((s.unidoConSimcity[i])$_0$).mapa).horizontales)))

casas(**in** s: simcity) $\rightarrow$ res : dicc(Pos, Nivel)

1: $res \leftarrow Vacio()$
2: $itCasas \leftarrow CrearIt(s.casas)$
3: $turnoAct \leftarrow s.turno$
4: **while** haySiguiente(itCasas) **do**
5: $nivel \leftarrow turnoAct - siguiente(itCasas).TurnoAparicion$
6: $definir(siguiente(itCasas).Pos, nivel, res)$
7: $avanzar(itCasas)$
8: **end while**
9: $itUniones \leftarrow CrearIt(s.unidoConSimcitys))$
10: **while** haySiguiente(itUniones) **do**
11: $itCasasUniones \leftarrow CreatIt(casas(siguiente(itUniones)).first)$
12: **while** haySiguiente(itCasasUniones) **do**
13: **if** !definido?(siguienteClave(itCasasUniones),res) **then**
14: $nivel \leftarrow (siguiente(itUniones).first).turno - siguienteSignificado(itCasasUniones) + turnoAct - (siguiente(itUniones)).second$
15: $definir(siguienteClave(itCasasUniones), nivel, res)$
16: **end if**
17: $avanzar(itCasasUniones)$
18: **end while**
19: $avanzar(itUniones)$
20: **end while**
21: **return** res

Complejidad: $O(Longitud(s.casas) + Longitud(s.unidoConSimcitys)$ x
22: $\sum_{i=0}^{Longitud(s.unidoConSimcitys)} \#claves(casas((s.unidoConSimcitys[i])_0)))$

comercios(**in** s: simcity) $\rightarrow$ res : dicc(Pos, Nivel)

1: $res \leftarrow Vacio()$
2: $itCom \leftarrow CrearIt(s.comercios)$
3: $turnoAct \leftarrow s.turno$
4: **while** haySiguiente(itCom) **do**
5: $nivel \leftarrow max(turnoAct - siguiente(itCom).TurnoAparicion, nivelComercio(\&s, siguiente(itCom).Pos))$
6: $definir(siguiente(itCom).Pos, nivel, res)$
7: $avanzar(itCom)$
8: **end while**
9: $itUniones \leftarrow CrearIt(s.unidoConSimcitys))$
10: **while** haySiguiente(itUniones) **do**
11: $itComUniones \leftarrow CreatIt(comercios(siguiente(itUniones)).first)$
12: **while** haySiguiente(itComUniones) **do**
13: **if** !definido?(siguienteClave(itComUniones),res) **then**
14: $nivel \leftarrow max((siguiente(itUniones).first).turno - siguienteSignificado(itComUniones) + turnoAct - (siguiente(itUniones)).second, nivelComercio(s, siguienteClave(itComUniones)))$
15: $definir(siguienteClave(itComUniones), nivel, res)$
16: **end if**
17: $avanzar(itComUniones)$
18: **end while**
19: $avanzar(itUniones)$
20: **end while**
21: **return** res

Complejidad: $O(Longitud(s.comercios) + Longitud(s.unidoConSimcitys)$ x
22: $\sum_{i=0}^{Longitud(s.unidoConSimcitys)} \#claves(comercios((s.unidoConSimcitys[i])_0))))$

nivelComercio(**in** *psc*: puntero(simcity), **in** *com*: Pos) → *res* : *Nat*

```
1: casas ← casas(psc)
2: itCasas ← CrearIt(casas)
3: mejor ← 1
4: while haySiguiente(itCasas) do
5:     if DistManhattan3(com, itCasas.siguienteClave) && (itCasas.siguiente).second >mejor then
6:         mejor ← (itCasas.siguiente).second
7:     end if
8:     avanzar(it)
9: end while
10: return mejor
```

Complejidad: $O(copy(casas(psc)))$

DistManhattan3(**in** *p*: Pos, **in** *q*: Pos) → *res* : bool

```
1: res ← (abs(p_0 − q_0) + abs(p_1 − q_1)) ≤ 3
2: return res
```

Complejidad: $O(1)$

---

popularidad(**in** *s*: simcity) → *res* : Nat

```
1: res ← s.popularidad
2: return res
```

Complejidad: $O(1)$

---

antigüedad(**in** *s*: simcity) → *res* : Nat

```
1: res ← s.antigüedad
2: return res
```

Complejidad: $O(1)$

---

turnos(**in** *s*: simcity) → *res* : Nat

```
1: res ← s.turnos
2: return res
```

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`, **in** *m* : `Mapa`)
Pre $\equiv$ {$sv_0 = sv \wedge$ n $\notin$ juegos(s)}
Post $\equiv$ {$sv =_{\text{obs}} registrar(sv_0, n, m)$}
Complejidad: $O(|Nombre| + copy(m))$
Descripción: Registra un nuevo simcity con el nombre dado

AVANZARTURNOSIMCITY(**in/out** *sv* : `servidor`, **in** *n* : `nombre`, **in** *dc* : `dicc(Pos,Construccion)`)
Pre $\equiv$ {$sv = sv_0 \wedge$ n $\in$ juegos(sv) $\wedge$ 1$\leq$#claves(dc) $\wedge_{\text{L}}$
($\forall p$: Pos)(p $\in$ claves(dc) $\Rightarrow_L$
p $\notin$ (claves(casas(sv, n)) $\cup$ claves(comercios(sv, n))) $\wedge$
p $\notin$ (horizontales(mapa(sv, n)) $\cup$ verticales(mapa(sv, n)))}
Post $\equiv$ {$sv =_{\text{obs}} avanzarTurnoSimcity(sv_0, n, dc)$}
Complejidad: $O(|Nombre| + \#claves(dc))$
Descripción: Avanza el turno de un simcity.

UNIRSIMCITYS(**in/out** *sv* : `servidor`, **in** *n*1 : `nombre`, **in** *n*2 : `nombre`) $\rightarrow$ *res* : `[`
Pre $\equiv$ {true}
Post $\equiv$ {}$sv = sv_0$ {n1,n2} $\subseteq$ juegos(sv) $\wedge$
//ríos no eliminan construcciones:
($\forall$ p : Pos)(p $\in$ (claves(casas(sv, n1)) $\cup$ claves(comercios(sv, n1))) $\Rightarrow$
($\pi_1(p)$ $\notin$ horizontales(mapa(sv, n2)) $\wedge$ $\pi_2(p)$ $\notin$ verticales(mapa(sv, n2)))) $\wedge$
($\forall$ p : Pos)(p $\in$ (claves(casas(sv, n2)) $\cup$ claves(comercios(simcity(sv, n2)))) $\Rightarrow$
($\pi_1(p)$ $\notin$ horizontales(mapa(sv, n1)) $\wedge$ $\pi_2(p)$ $\notin$ verticales(mapa(sv, n1))))

//no se pisan construcciones de nivel máximo:
vacio?(constrDeNivel(simcity(sv, n1), maxNivel(simcity(sv, n1))) $\cap$ (casas(sv, n2) $\cup$ comercios(sv, n2))) $\wedge$
vacio?(constrDeNivel(simcity(sv, n2), maxNivel(simcity(sv, n2))) $\cap$ (casas(sv, n1) $\cup$ comercios(sv, n1)))] sv
$=_{\text{obs}}$ unirSimcitys(sv_0, n1, n2)
Complejidad: $O(|Nombre|)$
Descripcion: Une dos simcitys.

MAPASIMCITY(**in** *sv* : `servidor`, **in** *n* : `Nombre`) $\rightarrow$ *res* : `Mapa`
Pre $\equiv$ {$n \in juegos(sv)$}
Post $\equiv$ {$res =_{\text{obs}} mapaSimcity(sv, n)$}
Complejidad: $O(|\text{Nombre}|)$ + O(mapa(significado (n, sv)))
Observacion: O(mapa(significado (n, sv))) es la complejidad calculada de mapa en el Módulo Simcity
Descripción: Devuelve el mapa del Simcity.

CASASSIMCITY(**in** *sv* : `servidor`, **in** *n* : `Nombre`) $\rightarrow$ *res* : `dicc(Pos, Nivel)`
Pre $\equiv$ {$n \in juegos(sv)$}
Post $\equiv$ {$res =_{\text{obs}} casasSimcity(sv, n)$}
Complejidad: $O(|\text{Nombre}|)$ + O(casas(significado (n, sv)))
Observacion: O(casas(significado (n, sv))) es la complejidad calculada de casas en el Módulo Simcity
Descripción: Devuelve las casas del Simcity.

COMERCIOSSIMCITY(**in** sv: `servidor`, **in** n: `Nombre`) → res : `dicc(Pos, Nivel)`
Pre ≡ $\{n \in juegos(sv)\}$
Post ≡ $\{res =_{\text{obs}} comerciosSimcity(sv, n)\}$
Complejidad: $O(|\text{Nombre}|)$ + O(comercios(significado (n, sv)))
Observacion: O(comercios(significado (n, sv))) es la complejidad calculada de comercios en el Módulo Simcity
Descripción: Devuelve los comercios del Simcity.

POPULARIDADSIMCITY(**in** sv: `servidor`, **in** n: `Nombre`) → res : `Nat`
Pre ≡ $\{n \in juegos(sv)\}$
Post ≡ $\{res =_{\text{obs}} popularidadSimcity(sv, n)\}$
Complejidad: $O(|Nombre|)$
Descripción: Devuelve la popularidad del Simcity.

ANTIGÜEDADSIMCITY(**in** sv: `servidor`, **in** n: `Nombre`) → res : `Nat`
Pre ≡ $\{n \in juegos(sv)\}$
Post ≡ $\{res =_{\text{obs}} antigüedadSimcity(sv, n)\}$
Complejidad: $O(|Nombre|)$
Descripción: Devuelve la antigüedad del Simcity.

TURNOSIMCITY(**in** sv: `servidor`, **in** n: `Nombre`) → res : `Nat`
Pre ≡ $\{n \in juegos(sv)\}$
Post ≡ $\{res =_{\text{obs}} turnoSimcity(sv, n)\}$
Complejidad: $O(|Nombre|)$
Descripción: Devuelve el turno actual del Simcity.

Representación

Representación de Servidor

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

dicc_trie(nombre, simcity)

dicc_trie es un diccionario con las mismas funcionalidades de un diccionario común, pero se almacena en un trie lo que hace que las complejidades de busqueda e inserción sean O(clave más larga).

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

Abs : estr s $\longrightarrow$ servidor {Rep(s)}
Abs(s) ≡ t : Servidor/ $claves(s) =_{\text{obs}} juegos(t) \wedge_{\text{L}}$
$(\forall n : nombre)(n \in juegos(t) \Rightarrow_L simcity(t, n) =_{\text{obs}} significado(n, s))$

Algoritmos

---

crearServidor()→ res : servidor
1: $res \leftarrow Vacio()$
2: **return**

Complejidad: $O(1)$

---

registrar(**in**/**out** sv: `servidor`, **in** n: `nombre`, **in** m: `mapa`)→ res : estr
1: $definir(n, iniciar(m), sv)$

Complejidad: $O(|Nombre| + copy(m))$

---

avanzarTurnoSimcity(**in/out** sv : `servidor`, **in** n : `nombre`, **in** dc : `dicc(Pos,Construccion)`)

1: $avanzarTurno(significado(n, sv))$

Complejidad: $O(|Nombre| + \#claves(dc))$

unirSimcitys(**in/out** sv : `servidor`, **in** $n1$: `nombre`, **in** $n2$: `nombre`)

1: $unir(significado(n1, sv), \&significado(n2, sv))$

Complejidad: $O(|Nombre|)$

mapaSimcity(**in** sv : `servidor`, **in** n : `nombre`)→ res : Mapa

1: **return** mapa(significado(n,sv))

Complejidad: $O(|\text{Nombre}|)$ + O(mapa(significado (n, sv))

casasSimcity(**in** sv : `servidor`, **in** n : `nombre`)→ res : dicc(Pos, Nivel)

1: **return** casas(significado(n,sv))

Complejidad: $O(|\text{Nombre}|)$ + O(casas(significado (n, sv))

comerciosSimcity(**in** sv : `servidor`, **in** n : `nombre`)→ res : dicc(Pos, Nivel)

1: **return** comercios(significado(n,sv))

Complejidad: $O(|\text{Nombre}|)$ + O(comercios(significado (n, sv))

popularidadSimcity(**in** sv : `servidor`, **in** n : `nombre`)→ res : Nat

1: **return** popularidad(significado(n,sv))

Complejidad: $O(|Nombre|)$

antigüedadSimcity(**in** sv : `servidor`, **in** n : `nombre`)→ res : Nat

1: **return** antigüedad(significado(n,sv))

Complejidad: $O(|Nombre|)$

tunoSimcity(**in** sv : `servidor`, **in** n : `nombre`)→ res : Nat

1: **return** turno(significado(n,sv))

Complejidad: $O(|Nombre|)$

2.4. Módulo Dicc_Trie

Interfaz

se explica con: DICC(CLAVE, SIGNIFICADO)

géneros: `dicc_trie`

Operaciones básicas de dicc_trie

VACÍO() → *res* : `dicc_trie(`κ`, `σ`)`
Pre ≡ {true}
Post ≡ {res $=_{obs}$ vacio(m)}
Complejidad: $O(1)$
Descripción: genera un diccionario vacío.

DEFINIR(**in/out** d: `dicc_trie(`κ`, `σ`)`, **in** k: κ, **in** s: σ)
Pre ≡ {d $=_{\text{obs}}$ d_0}
Post ≡ {d $=_{obs}$ definir(d,k,s)}
Complejidad: $O(|\text{Clave más larga}|)$
Descripción: define la clave k con el significado s en el diccionario.
Aliasing: Los elementos k y s se definen por copia.

DEFINIDO?(**in** d: `dicc_trie(`κ`, `σ`)`, **in** k: κ) → *res* : `bool`
Pre ≡ {true}
Post ≡ {res $=_{obs}$ def?(d,k)}
Complejidad: $O(|\text{Clave más larga}|)$
Descripción: devuelve true si y sólo k está definido en el diccionario.

SIGNIFICADO(**in** d: `dicc_trie(`κ`, `σ`)`, **in** k: κ) → *res* : σ
Pre ≡ {def?(d,k)}
Post ≡ {res $=_{obs}$ significado(d,k)}
Complejidad: $O(|\text{Clave más larga}|)$
Descripción: devuelve el significado de la clave k en d.
Aliasing: res es modificable si y sólo si d es modificable.