DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

TP de Interfaz

Sim City

10 de mayo de 2020 — Algoritmos y Estructuras de Datos II

Grupo 6

Integrante	LU	Correo electrónico
Silva Fernandez, Ignacio Tomas	410/19	ignaciotomas.silva622@gmail.com
Damburiarena, Gabriel	889/19	gabriel.damburiarena@gmail.com
Guastella, Mariano	888/19	marianoguastella@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

1. Renombres

Casilla : tupla<Nat, Nat>
Direccion : String
Posicion : Nat
Nivel : Nat

2. Módulo Mapa

Interfaz

se explica con: MAPA

géneros: `mapa`

Operaciones básicas de mapa

CREAR() $\rightarrow$ *res* : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} nuevoMapa()$}
Complejidad: $O(1)$
Descripción: Crea un mapa vacío.

AGREGARRIO(**in/out** m: `mapa`, **in** D: `direccion`, **in** p: `posicion`)
Pre $\equiv$ {m=m0}
Post $\equiv$ {$m =_{\text{obs}} agregarRio(m0, d, p)$}
Complejidad: $O(1)$
Descripción: Agrega un río al mapa.

HAYRIO(**in** m: `mapa`, **in** c: `casilla`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} hayRio(m, c)$}
Complejidad: $O(max(\#(estr.horizontales), \#(estr.verticales)))$
Descripción: Determina si un río pertenece al mapa.

UNIRMAPA(**in/out** $m1$: `mapa`, **in** $m2$: `mapa`) $\rightarrow$ *res* : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} unirMapa(m1, m2)$}
Complejidad: $O(copy(m2))$
Descripción: Une dos mapas en uno.
Aliasing: El mapa m2 se pasa por copia.

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. Representamos los conjuntos lineales con listas enlazadas, de manera que el operador ++ es concatenar dos listas enlazadas que toma el ultimo nodo y lo une con el primero de la otra lista, esto tiene costo O(1).

`mapa` **se representa con** `estr`

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

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

Abs : estr m $\longrightarrow$ mapa {Rep(m)}
Abs(m) $\equiv$ horizontales(m) = $estr$.horizontales $\wedge$ verticales(m) = $estr$.verticales

Algoritmos

Para abreviar tupla<conj_lineal(Nat), conj_lineal(Nat)>= estr.

---

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

1: $res.horizontales \leftarrow \emptyset$
2: $res.verticales \leftarrow \emptyset$

Complejidad: $O(1)$

---

iAgregarRio(**in/out** m : `estr`, **in** d : `string`, **in** p : `Nat`)

1: **if** d = horizontal **then** m.horizontales.agregar(p) **else** m.verticales.agregar(p) **fi**

Complejidad: $O(1)$

---

iHayRio(**in** m : `estr`, **in** c : `casilla`) $\rightarrow$ *res* : bool

1: $res \leftarrow (\pi_1(c) \in m.verticales \vee \pi_2(c) \in m.horizontales)$
2: **return** res

Complejidad: $O(n)$ *siendo n la cantidad de elementos del conjunto*

---

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

1: $m1.horizontales \leftarrow m1.horizontales + +m2.horizontales$
2: $m1.verticales \leftarrow m1.verticales + +m2.verticales$

Complejidad: $O(copy(m2.verticales) + copy(m2.horizontales))$

---

3. Módulo SimCity

Interfaz

se explica con: SimCity

géneros: `SimCity`

Operaciones básicas de SimCity

empezarPartida(**in** m : `mapa`) $\rightarrow$ res : `SimCity`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} empezarPartida(m)\}$
Complejidad: $O(1)$
Descripción: Crea un SimCity vacío con el mapa dado.

agregarCasa(**in/out** s : `SimCity`, **in** c : `Casilla`)
Pre $\equiv$ $\{sePuedeConstruir(s, c) \land s = s0\}$
Post $\equiv$ $\{s =_{\text{obs}} agregarCasa(s0, c)\}$
Complejidad: $O(1)$
Descripción: Agrega una casa al SimCity si la casilla no está ocupada.
Aliasing: La casilla c se pasa por copia.

agregarComercio(**in/out** s : `SimCity`, **in** c : `Casilla`)
Pre $\equiv$ $\{sePuedeConstruir(s, c) \land s = s0\}$
Post $\equiv$ $\{s =_{\text{obs}} agregarComercio(s0, c)\}$
Complejidad: $O(1)$
Descripción: Agrega un comercio al SimCity si la casilla no está ocupada.
Aliasing: La casilla c se pasa por copia.

avanzarTurno(**in/out** s : `SimCity`)
Pre $\equiv$ $\{huboConstruccion(s) \land s = s0\}$
Post $\equiv$ $\{s =_{\text{obs}} avanzarTurno(s0)\}$
Complejidad: $O(n)$ *siendo n el maximo entre la cantidad de casas y comercios*
Descripción: Pasa al siguiente turno si ya se construyó en el actual.

unir(**in/out** $s1$: `SimCity`, **in** $s2$: `SimCity`)
Pre $\equiv$ $\{(\forall c$: Casilla$)c \in construcciones(s1) \Rightarrow \neg hayRio(mapa(s2), c) \land_{\text{L}}$
$(\forall c$: Casilla$)c \in construccion(s2) \Rightarrow \neg hayRio(mapa(s1), c) \land_{\text{L}}$
$(\forall c1, c2$: Casilla$)(c1 \in construcciones(s1) \lor c2 \in construcciones(s2)) \Rightarrow$
$(esCasillaDeMaximoNivel(s1, c1) \lor esCasillaDeMaximoNivel(s2, c2) \Rightarrow c1 \neq c2)) \land s1 = s0\}$
Post $\equiv$ $\{s1 =_{\text{obs}} unir(s0, s2)\}$
Complejidad: $O(1)$
Descripción: Une dos partidas de SimCity, siempre y cuando no haya rios solapando construcciones entre los mapas. Tampoco pisa las construcciones de nivel maximo de cada mapa. Para la union prioriza a la propiedad con nivel mayor. 1 [El SimCity s2 se pasa por copia.]

mapa(**in** s : `SimCity`) $\rightarrow$ res : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} mapa(s)\}$
Complejidad: $O(n^2)$ *siendo n la cantidad de elementos del conjunto*
Descripción: Devuelve el mapa del simcity dado. Modifica el simcity sacando los repetidos.
Aliasing: El SimCity s se pasa por referencia.

casas(**in** s : `SimCity`) $\rightarrow$ res : `conj(casilla)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} casas(s)\}$
Complejidad: $O(n^2)$ *siendo n la cantidad de elementos del conjunto*
Descripción: Devuelve las casas de un SimCity. Modifica el simcity sacando los repetidos.
Aliasing: El SimCity s se pasa por referencia.

COMERCIOS(**in** s : `SimCity`) $\rightarrow$ res : `conj(casilla)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} comercios(s)\}$
Complejidad: $O(n^2)$ *siendo n la cantidad de elementos del conjunto*
Descripción: Devuelve los comercios de un SimCity. Modifica el simcity sacando los repetidos.
Aliasing: El SimCity s se pasa por referencia.

NIVEL(**in** s : `SimCity`, **in** c : `Casilla`) $\rightarrow$ res : `Nat`
Pre $\equiv$ $\{hayContruccion(s, c)\}$
Post $\equiv$ $\{res =_{\text{obs}} nivel(s, c)\}$
Complejidad: $O(card)$
Descripción: Devuelve el nivel de la construccion en la casilla c, si hay una construccion en esa casilla.
Aliasing: La casilla c se pasa por copia.

HUBOCONSTRUCCION(**in** s : `SimCity`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} huboConstruccion(s)\}$
Complejidad: $O(1)$
Descripción: Devuelve si se construyo en este turno en el SimCity.

POPULARIDAD(**in** s : `SimCity`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} popularidad(s)\}$
Complejidad: $O(1)$
Descripción: Devuelve cuantas uniones hubo en esta partida de SimCity.

ANTIGUEDAD(**in** s : `SimCity`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} antiguedad(s)\}$
Complejidad: $O(1)$
Descripción: Devuelve la cantidad de turnos que se pasaron en esa partida.

CONSTRUCCIONES(**in** s : `SimCity`) $\rightarrow$ res : `Conj_lineal(Casilla)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} construcciones(s)\}$
Complejidad: $O(n^2)$ *siendo n la cantidad de elementos del conjunto*
Descripción: Devuelve todas las contrucciones del SimCity dado.

HAYCONSTRUCCION(**in** s : `SimCity`, **in** c : `Casilla`) $\rightarrow$ res : `Bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} hayConstruccion(s, c)\}$
Complejidad: $O(n^2)$ *siendo n la cantidad de elementos del conjunto*
Descripción: Indica si la casilla está ocupada con una construcción en el SimCity dado.

SEPUEDECONSTRUIR(**in** s : `SimCity`, **in** c : `Casilla`) $\rightarrow$ res : `Bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} sePuedeConstruir(s, c)\}$
Complejidad: $O(n^2)$ *siendo n la cantidad de elementos del conjunto*
Descripción: Indica si la casilla está ocupada con un rio o con una construcción en el SimCity dado.

ESCASILLADEMAXIMONIVEL(**in** s : `SimCity`, **in** c : `Casilla`) $\rightarrow$ res : `Bool`
Pre $\equiv$ $\{hayConstruccin(s, c)\}$
Post $\equiv$ $\{res =_{\text{obs}} esCasaDeMaximoNivel(s, c)\}$
Complejidad: $o(n^2)$ *siendo n la cantidad de elementos del conjunto*
Descripción: Indica si la casilla está ocupada con una construcción de máximo nivel en el SimCity dado.

MAXIMONIVEL(**in** s : `SimCity`, **in** sc : `Conj(Casilla)`) $\rightarrow$ res : `Nat`
Pre $\equiv$ $\{sc = construcciones(s)\}$
Post $\equiv$ $\{res =_{\text{obs}} maximoNivel(s, sc)\}$
Complejidad: $O(1)$
Descripción: Devuelve el nivel máximo de s.

Aliasing: El conjunto de casillas sc se pasa por copia.

DARNIVELACOMERCIO(**in** $s1$: `SimCity`, **in** $s2$: `SimCity`, **in** c : `Casilla`) $\rightarrow$ res : `Nat`
Pre $\equiv$ $\{c \in comercios(s1) \cup comercios(s2)\}$
Post $\equiv$ $\{res =_{\text{obs}} darNivelAComercio(s1, s2, c)\}$
Complejidad: $O(n)$ *siendo n la cantidad de elementos del conjunto*
Descripción: Devuelve el nivel que le corresponde al comercio c.
Aliasing: La casilla c se pasa por copia.

DISTANCIAMANHATTAN(**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 manhattan entre las casillas x e y.
Aliasing: Las casillas x e y se pasan por copia.

Representación

Representación de SimCity
Representamos los conjuntos lineales con listas enlazadas, de manera que el operador ++ es concatenar dos listas enlazadas que toma el ultimo nodo y lo une con el primero de la otra lista, esto tiene costo O(1).

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

donde `estr` es `tupla`(*mapa*: `mapa`
, *casas*: `conj_lineal(<Casilla, Nivel>)`
, *comercios*: `conj_lineal(<Casilla, Nivel>)`
, *turno*: `Nat`
, *popularidad*: `Nat`
, *huboConstruccion*: `bool`
, *uniones*: `conj_lineal(SimCity)`)

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\iff$ e.turno = e.MaximoNivel(e, e.casas $\cup$ e.comercios) $\wedge$
($\forall$ c $\in$ e.casas)($\pi_1(casilla(c)) \notin$ mapa.horizontales $\wedge$ $\pi_2(casilla(c)) \notin$ mapa.verticales $\wedge$ c $\notin$ e.comercios) $\wedge$
($\forall$ c $\in$ e.comercios)($\pi_1(casilla(c) \notin$ mapa.horizontales $\wedge$ $\pi_2(casilla(c) \notin$ mapa.verticales) $\wedge$
($\forall$ s $\in$ e.uniones)(Rep(e))

Abs : estr s $\longrightarrow$ SimCity {Rep(s)}
Abs(s) $\equiv$ s : SimCity |
mapa(s) = UnirConjSimcity(s, e.uniones).mapa $\wedge$
casas(s) = UnirConjSimcity(s, e.uniones).casas $\wedge$
comercios(s) = UnirConjSimcity(s, e.uniones).comercios $\wedge$
popularidad(s) = UnirConjSimcity(s, e.uniones).popularidad $\wedge$
huboConstruccion(s) = UnirConjSimcity(s, e.uniones).huboConstruccion $\wedge$
nivel(s, c) = buscarNivel(UnirConjSimcity(s, e.uniones).casas $\cup$ UnirConjSimcity(s, e.uniones).comercios, c)

$\forall$ cj: conj(SimCity) s: SimCity
buscarNivel(props, c) $\equiv$ **if** c = π_1(dameUno(props)) **then** π_2(dameUno(props)) **else** buscarNivel(sinUno(props), c) **fi**

UnirConjSimcity(s,cj) $\equiv$ **if** c = $\emptyset$ **then** s **else** UnirConjSimcity(Unir(s, dameUno(cj)), sinUno(cj)) **fi**

Algoritmos

Para abreviar estr = <mapa : mapa,
casas : conj_lineal(<Casilla, Nivel>),
comercios : conj_lineal(<Casilla, Nivel>),

turno : Nat,
popularidad :Nat,
huboConstruccion : bool,
uniones : conj_lineal(SimCity)>.

---

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

1: $res.mapa \leftarrow m$
2: $res.casas \leftarrow \emptyset$
3: $res.comercios \leftarrow \emptyset$
4: $res.turno \leftarrow 0$
5: $res.popularidad \leftarrow 0$
6: $res.huboConstruccion \leftarrow false$
7: $res.uniones \leftarrow \emptyset$

Complejidad: $O(1)$

---

iAgregarCasa(**in/out** e : `estr`, **in** c : `casilla`)

1: $Ag(c,\ e.casas)$
2: $e.huboConstrucciones \leftarrow true$

Complejidad: $O(1)$

---

iAgregarComercio(**in/out** e : `estr`, **in** c : `casilla`))

1: $Ag(c,\ e.comercios)$
2: $e.huboConstrucciones \leftarrow true$

Complejidad: $O(1)$

---

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

1: $e.turno \leftarrow e.turno + 1$
2: **for** casa in e.casas **do**
3: casa.nivel ← casa.nivel + 1
4: **end for**
5: **for** comercio in e.comercio **do**
6: comercio.nivel ← comercio.nivel + 1
7: **end for**

Complejidad: $O(n)$ siendo n el maximo entre la cantidad de casas y la cantidad de comercios.

---

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

1: $e1.turno \leftarrow max(e1.turno, e2.turno)$
2: $e1.popularidad \leftarrow e1.popularidad + e2.popularidad + 1$
3: $e1.huboConstruccion \leftarrow e1.huboConstruccion$
4: $Agregar(e1.uniones, e2)$

Complejidad: $O(1)$

---

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

1: res $\leftarrow <\emptyset, \emptyset>$
2: **for** simcity in e.uniones **do** ▷ n por la complejidad del ciclo
3: $e.mapa + +simcity.mapa$ ▷ O(1)
4: **end for** ▷ O(n) * O(1)
5: **for** *coordenada in* $\pi_1(e.mapa)$ **do** ▷ es n por la complejidad del ciclo
6: **if** coordenada $\notin \pi_1(res)$ **then** ▷ n ya que $\in$ es lineal
7: agregar(coordenada, $\pi_1(res)$) ▷ O(1)
8: **end if**
9: **end for** ▷ O(n) * O(n) = $O(n^2)$
10: **for** *coordenada in* $\pi_2(e.mapa)$ **do** ▷ es n por la complejidad del ciclo
11: **if** coordenada $\notin \pi_2(res)$ **then** ▷ n ya que $\in$ es lineal
12: agregar(coordenada, $\pi_2(res)$) ▷ O(1)
13: **end if**
14: **end for** ▷ O(n) * O(n) = $O(n^2)$
15: $e.mapa \leftarrow res$
16: **return** res

Complejidad: $O(n^2)$

---

iCasas(**in** e : `estr`) $\rightarrow$ res : conj_lineal(<Casilla, Nivel>)

1: **for** simcity in e.uniones **do** ▷ n por la complejidad del ciclo
2: $e.casas + +simcity.casas$ ▷ O(1)
3: **end for** ▷ O(n) * O(1)
4: $res \leftarrow \emptyset$
5: **for** casa1 in e.casas **do** ▷ n por la complejidad del ciclo
6: **for** casa2 in res **do** ▷ n por la complejidad del ciclo
7: **if** casa1.casilla = casa2.casilla **then**
8: casa2.nivel = max(casa1.nivel, casa2.nivel) ▷ O(1)
9: **else**
10: agregar(casa1, res) ▷ O(1)
11: **end if**
12: **end for** ▷ O(n) * O(1)
13: **end for** ▷ O(n) * O(n) = $O(n^2)$
14: e.casas $\leftarrow$ res **return** res

Complejidad: $O(n^2)$

iComercios(**in** e : `estr`) → res : conj_lineal(<Casilla, Nivel>)

```
1:  resAux ← ∅
2:  for simcity in e.uniones do                                   ▷ n por la complejidad del ciclo
3:      e.comercios + +simcity.comercios                          ▷ O(1)
4:  end for                                                       ▷ O(n) * O(1)
5:  for comercio1 in e.comercios do                               ▷ n por la complejidad del ciclo
6:      for comercio2 in resAux do                                ▷ n por la complejidad del ciclo
7:          if comercio1.casilla = comercio2.casilla then
8:              comercio2.nivel = max(comercio1.nivel, comercio2.nivel)   ▷ O(1)
9:          else
10:             agregar(comercio1, resAux)                        ▷ O(1)
11:         end if
12:     end for                                                   ▷ O(n) * O(1)
13: end for                                                       ▷ O(n) * O(n) = O(n^2)
14: res ← ∅
15: for comercio in resAux do                                     ▷ n por la complejidad del ciclo
16:     encontrado ← false                                        ▷ O(1)
17:     for casa in iCasas(e) do                                  ▷ n por la complejidad del ciclo
18:         if comercio.casilla = casa.casilla then
19:             encontrado ← true                                 ▷ O(1)
20:         end if
21:     end for                                                   ▷ O(n) * O(1)
22:     if ¬encontrado then
23:         agregar(comercio, res)
24:     end if
25: end for                                                       ▷ O(n) * O(n) = O(n^2)
26: for comercio in res do                                        ▷ n por la complejidad del ciclo
27:     comercio.Nivel ← iDarNivelComercios(e, comercio.casilla)
28: end for                                                       ▷ O(n) * O(1)
29: e.comercios ← res return res
```

Complejidad: $O(n^2)$

---

iNivel(**in** e : `estr`, **in** c : `Casilla`) → res : Nat

```
1:  if pertenece(iCasas(e), c) then
2:      for casa in iCasas(e) do          ▷ este ciclo se ejecuta n veces si n es la cantidad de casas
3:          if π1(casa) == c then
4:              res ← π2(casa)
5:          end if
6:      end for                           ▷ Complejidad total: O(n)
7:  else
8:      for comercio in iComercios(e) do  ▷ este ciclo se ejecuta n veces si n es la cantidad de comercios
9:          if π1(comercio) == c then
10:             res ← π2(comercio)
11:         end if
12:     end for                           ▷ Complejidad total O(n)
13: end if
```

Complejidad: $O(n)$

---

iHuboConstruccion(**in** e : `estr`) → res : bool

```
1:  res ← e.huboConstruccion
```

Complejidad: $O(1)$

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

1: $res \leftarrow e.popularidad$

Complejidad: $O(1)$

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

1: $res \leftarrow e.turno$

Complejidad: $O(1)$

iConstrucciones(**in** e : `estr`) $\rightarrow$ res : conj_lineal(<Casilla, Nivel>)

1: $res \leftarrow iCasas(e) \cup iComercios(e)$

Complejidad: $O(n^2)$

iHayConstruccion(**in** e : `estr`, **in** c : `Casilla`) $\rightarrow$ res : bool

1: $res \leftarrow pertenece(iConstrucciones(e), c)$

Complejidad: $O(n^2)$

iSePuedeConstruir(**in** e : `estr`, **in** c : `Casilla`) $\rightarrow$ res : bool

1: $res \leftarrow \neg iHayConstruccion(e, c) \wedge \neg ihayRio(iMapa(e), c)$

Complejidad: $O(n^2)$

iEsCasillaDeMaximoNivel(**in** e : `estr`, **in** c : `Casilla`) $\rightarrow$ res : bool

1: $res \leftarrow iNivel(e, c) = iMaximoNivel(e, iConstrucciones(e))$

Complejidad: $O(n^2)$

iMaximoNivel(**in** e : `estr`, **in** sc : `conj_lineal(<Casilla, Nivel>)`) $\rightarrow$ res : Nat

1: $res \leftarrow e.turno$

Complejidad: $O(1)$

iDarNivelAComercio(**in** e : `estr`, **in** c : `Casilla`) $\rightarrow$ res : Nat

1: $res \leftarrow 0$
2: **for** casa in iCasas(e) **do**
3: **if** $iDistanciaManhattan(pi_1(casa), c) \leq 3 \wedge \pi_2(casa) > res$ **then**
4: $res \leftarrow \pi_2(casa)$
5: **end if**
6: **end for**

Complejidad: $O(n)$

iDistanciaManhattan(**in** x : `Casilla`, **in** y : `Casilla`) $\rightarrow$ res : Nat

1: $res \leftarrow |\pi_1(x) - \pi_1(y)| + |\pi_2(x) - \pi_2(y)|$

Complejidad: $O(1)$