DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Simcity

Trabajo practico grupal 3

19 de junio de 2020 — Algoritmos y Estructuras de Datos II

Grupo 17

Integrante	LU	Correo electrónico
Rodriguez Celma, Guido	374/19	guido.rodriguez@outlook.com.ar
Itzcovitz, Ryan	169/19	ryanitzcovitz@gmail.com
Rodriguez, Miguel	57/19	mmiguerodriguez@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
https://exactas.uba.ar

1. Módulo Mapa

Interfaz

se explica con: MAPA

géneros: `mapa`

Operaciones básicas de mapa

CREAR(**in** hs: `conj(Nat)`, **in** vs: `conj(Nat)`) $\rightarrow$ res : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} agregarRios(nuevoMapa(), hs, vs))\}$
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: $O(\#(estr.horizontales) + \#(estr.verticales))$
Descripción: Devuelve true si la casilla está ocupada por un río

UNIRMAPA(**in** $m1$: `mapa`, **in/out** $m2$: `mapa`)
Pre $\equiv$ $\{m2 = m2_0\}$
Post $\equiv$ $\{m2 =_{\text{obs}} unirMapa(m1, m2_0)\}$
Complejidad: Complejidad: O(#(m1.horizontales) $\times$ #(m2.horizontales) + #(m1.verticales) $\times$ #(m1.verticales))
Descripción: Une dos mapas

Representación

Representación de mapa

Un mapa contiene ríos infinitos horizontales y verticales. Los ríos 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 $\{\text{Rep}(m)\}$
Abs(m) $\equiv$ agregarRio(nuevoMapa(), estr.horizontales, estr.verticales)

Algoritmos

---

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

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

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

---

iHayRio?(**in/out** m: `estr`, **in** c: `Casilla`) $\rightarrow$ res : bool

1: **return** $pertenece?(m.horizontales, \pi_1(c)) \vee pertenece?(m.verticales, \pi_2(c))$

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

---

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

1: $itHor \leftarrow crearIt(m1.horizontales)$
2: **while** HaySiguiente?(itHor) **do**
3: $\quad agregar(m2.horizontales, Siguiente(itHor))$
4: $\quad Avanzar(itHor)$
5: **end while**
6: $itVer \leftarrow crearIt(m1.verticales)$
7: **while** HaySiguiente?(itVer) **do**
8: $\quad agregar(m2.verticales, Siguiente(itVer))$
9: $\quad Avanzar(itVer)$
10: **end while**

Complejidad: O(#(m1.horizontales) × #(m2.horizontales) + #(m1.verticales) × #(m1.verticales))

2. Módulo Simcity

Interfaz

se explica con: SIMCITY

géneros: `simCity`

Operaciones básicas de simCity

EMPEZAR(**in** m : `mapa`) $\rightarrow$ res : `simCity`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} empezarPartida(m)\}$
Complejidad: $O(copy(m))$
Descripción: Empezar la partida del SimCity

AGREGARCASA(**in/out** s : `SimCity`, **in** c : `Casilla`)
Pre $\equiv$ $\{s = s_0 \wedge_{\text{L}} seConstruir(s, c)\}$
Post $\equiv$ $\{s =_{\text{obs}} agregarCasa(s_0, c)\}$
Complejidad: $O(1)$
Descripción: Agrega una casa al SimCity

AGREGARCOMERCIO(**in/out** s : `SimCity`, **in** c : `Casilla`)
Pre $\equiv$ $\{s = s_0 \wedge_{\text{L}} seConstruir(s, c)\}$
Post $\equiv$ $\{s =_{\text{obs}} agregarComercio(s_0, c)\}$
Complejidad: $O(1)$
Descripción: Agrega un comercio al SimCity

NUEVOTURNO(**in/out** s : `simCity`) $\rightarrow$ res : `bool`
Pre $\equiv$ $\{s = s_0\}$
Post $\equiv$ $\{res = huboConstruccion(s) \wedge_{\text{L}} (res \Rightarrow_{\text{L}} s =_{\text{obs}} avanzarTurno(s_0)) \wedge (\neg res \Rightarrow s = s_0)\}$
Complejidad: Recorrer todas las construcciones
Descripción: Avanzar el turno del SimCity

UNIR(**in/out** $s1$: `simCity`, **in** $s2$: `simCity`)
Pre $\equiv$ $\{s1 = s1_0 \wedge (\forall c$: Casilla)$(c \in$ construcciones(s1) $\Rightarrow_{\text{L}} \neg$ hayRio(mapa(s2), c)) $\wedge_{\text{L}}$ (∀ c: Casilla)(c ∈ construcciones(s2) $\Rightarrow_{\text{L}} \neg$ hayRio(mapa(s1), c)) $\wedge_{\text{L}}$ (∀ c2, c2: Casilla)(c1 ∈ construcciones(s1) ∧ c2 ∈ construcciones(s2) $\Rightarrow_{\text{L}}$ (esCasillaMaximoNivel(s1, c1) ∨ esCasillaMaximoNivel(s2, c2) $\Rightarrow$ c1 ≠ c2))$\}$
Post $\equiv$ $\{s1 =_{\text{obs}} unir(s1_0, s2_0)\}$
Complejidad: $O(1)$
Descripción: Unir dos partidas de SimCity

NIVEL(**in** s: `simCity`, **in** c: `casilla`, **out** n: `nat`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = hayConstruccion(s, c) \wedge_{\text{L}} (res \Rightarrow_{\text{L}} n =_{\text{obs}} nivel(s, c))\}$
Complejidad: $O(\#(comercios(s)) + \#(casas(s)))$
Descripción: Nivel de la construcción en la casilla c

APLANARCASAS(**in** s: `simCity`) $\rightarrow$ res : `conj(casilla)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = casas(s)\}$
Complejidad: Agarrar las casas del simCity actual, con recursion agarrar las casas de máxima antigüedad por cada casillero de sus uniones, iterar sobre ellos y guardar sus casillas
Descripción: Retorna todas las casillas donde hay casas

APLANARCOMERCIOS(**in** s: `simCity`) $\rightarrow$ res : `conj(casilla)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = comercios(s)\}$
Complejidad: Agarrar las comercios del simCity actual, con recursion agarrar las comercios de máxima antigüedad por cada casillero de sus uniones, iterar sobre ellos y guardar sus casillas
Descripción: Retorna todas las casillas donde hay comercios

Representación

Representación de SimCity

Un simCity tiene distintas propiedades que dependen de su valor en la estructura como por ejemplo turnoActual, popularidad y huboConstrucción que en el caso de estar en una unión dejan de tener validez. En cambio, las casas, comercios y ríos no son representados únicamente por los que se encuentran en su estructura particular (e.casas, e.comercios y e.ríos) sino que es la suma de estas mismas propiedades para todas sus uniones. Cuando hablamos de estas tres variables en un simCity, incluimos también la unión de las mismas con distintos simCity's (los que se encuentran en e.uniones).

`construccion` **se representa con** `constr`

donde `constr` es `tupla`(*casilla*: `tupla<nat, nat>` , *antigüedad*: `nat`)

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

donde `estr` es `tupla`(*rios*: `mapa` , *turnoActual*: `nat`, *uniones*: `conjLineal<simCity>` , *casas*: `conjLineal<constr>` , *comercios*: `conjLineal<constr>` , *popularidad*: `nat` , *huboConstruccion*: `bool`)

Invariante de representación en lenguaje natural

1. No hay casas y comercios que estén en el mismo casillero.
2. No hay un río en los casilleros donde hay casas o comercios.
3. Existe una casa o comercio tal que su antigüedad es igual a turnoActual y no existe otra que su antigüedad sea mayor a turnoActual.
4. La popularidad es la cantidad de uniones totales.

Función de abstracción en lenguaje natural

1. mapa(s) = iAplanarRios(e)
2. casas(s) = iAplanarCasas(e) (conjunto de todos los casilleros donde hay una casa)
3. comercios(s) = iAplanarComercios(e) (conjunto de todos los casilleros donde hay un comercio)
4. nivel(s, c) = iNivel(e, c) (para toda construcción c en iAplanarCasas(e) $\cup$ iAplanarComercios(e) de un simCity s)
5. huboConstrucción(s) = e.huboConstrucción
6. popularidad(s) = e.popularidad

7. antigüedad(s) = e.turnoActual

Algoritmos

---

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

1: $res.rios \leftarrow m$
2: $res.casas \leftarrow Vacio()$
3: $res.comercios \leftarrow Vacio()$
4: $res.uniones \leftarrow Vacio()$
5: $res.huboConstruccion \leftarrow false$
6: $res.turnoActual \leftarrow 0$
7: $res.popularidad \leftarrow 0$
8: **return** estr

Complejidad: O(copy(m))

---

iAgregarCasa(**in/out** s : `estr`, **in** c : `Casilla`)

1: $agregarRapido(s.casas, < c, 0 >)$
2: $s.huboConstruccion \leftarrow true$

Complejidad: O(1)

---

iAgregarComercio(**in/out** s : `estr`, **in** c : `Casilla`)

1: agregarRapido(s.comercios, <c, 0>)
2: $s.huboConstruccion \leftarrow true$

Complejidad: O(1)

---

iNuevoTurno(**in**/**out** s : `estr`)

```
1: if s.huboConstruccion = true then
2:     itCasas ← CrearIt(s.casas)
3:     while HaySiguiente?(itCasas) do
4:         casa ← Siguiente(itCasas)
5:         π2(casa) ← π2(casa) + 1
6:         Avanzar(itCasas)
7:     end while
8:     itComercios ← CrearIt(s.comercios)
9:     while HaySiguiente?(itComercios) do
10:        comercio ← Siguiente(itComercios)
11:        π2(comercio) ← π2(comercio) + 1
12:        Avanzar(itComercios)
13:    end while
14:    itUniones ← CrearIt(s.uniones)
15:    while HaySiguiente?(itUniones) do
16:        s' ← siguiente(itUniones)
17:        s'.huboConstruccion ← true
18:        nuevoTurno(s')
19:        Avanzar(itUniones)
20:    end while
21:    s.turnoActual ← s.turnoActual + 1
22:    s.huboContruccion ← false
23: end if
```

Complejidad: Es el costo de recorrer todas las construcciones

---

iAplanarCasas(**in** s : `simCity`) → res : conj(casilla)

```
1: casas ← Vacio()
2: itCasas ← CrearIt(casas(s))
3: while HaySiguiente?(itCasas) do
4:     casa ← Siguiente(itCasas)
5:     agregarRapido(casas, casa.casilla)
6:     Avanzar(itCasas)
7: end while
8: return casas
```

Complejidad: O(casas(s))

---

iAplanarComercios(**in** s : `simCity`) → res : conj(casilla)

```
1: comercios ← Vacio()
2: itComercios ← CrearIt(comercios(s))
3: while HaySiguiente?(itComercios) do
4:     comercio ← Siguiente(itComercios)
5:     agregarRapido(comercios, comercio.casilla)
6:     Avanzar(itComercio)
7: end while
8: return comercio
```

Complejidad: O(comercios(s))

---

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

```
1: itUniones ← CrearIt(s.uniones)
2: while HaySiguiente?(itUniones) do
3:     union ← Siguiente(itUniones)
4:     unirMapa(aplanarRios(union), s.mapa)
5:     Avanzar(itCasas)
6: end while
7: return s.mapa
```

Complejidad: Cantidad total de uniones $\times$ el costo de hacer unirMapa (varia dependiendo de la cantidad de ríos horizontales y verticales de cada simCity)

---

iCasas(**in** s : `simCity`) $\rightarrow$ res : conj(constr)

```
1: casas ← Vacio()
2: itCasas ← CrearIt(s.casas)
3: while HaySiguiente?(itCasas) do
4:     agregarRapido(casas, Siguiente(itCasas))
5:     Avanzar(itCasas)
6: end while
7: itUniones ← CrearIt(s.uniones)
8: while HaySiguiente?(itUniones) do
9:     itUnionesCasas ← CrearIt(casas(Siguiente(itUniones)))
10:    while HaySiguiente?(itUnionesCasas) do
11:        tmpCasas ← comercios(Siguiente(itUnionesCasas))
12:        itCasasUnion ← CrearIt(tmpCasas)
13:        while HaySiguiente?(itCasasUnion) do
14:            casaUnion ← Siguiente(itCasasUnion)
15:            esta? ← false
16:            itCasasRes ← CrearIt(casas)
17:            while HaySiguiente?(itCasasRes) do
18:                casaRes ← Siguiente(itCasasRes)
19:                if (casaUnion.casilla = casaRes.casilla) then
20:                    esta? ← true
21:                    if (casaUnion.antigedad > casaRes.antigedad) then
22:                        casaRes.antigedad ← casaUnion.antigedad
23:                    end if
24:                end if
25:                Avanzar(itCasas)
26:            end while
27:            if (¬esta?) then
28:                agregarRapido(casas, casaUnion)
29:            end if
30:            Avanzar(itCasas)
31:        end while
32:        agregar(casas, tmpCasas)
33:        Avanzar(itUnionesCasas)
34:    end while
35:    Avanzar(itUniones)
36: end while
37: return casas
```

Complejidad: Es el costo de agarrar las casas del simCity actual, con recursion agarrar las casas de máxima antigüedad por cada casillero de sus uniones e iterar sobre ellos

iComercios(**in** s : `simCity`) $\rightarrow$ res : conj(constr)

```
1: comercios ← Vacio()
2: itComercios ← CrearIt(s.comercios)
3: while HaySiguiente?(itComercios) do
4:     agregarRapido(comercios, Siguiente(itComercios))
5:     Avanzar(itComercios)
6: end while
7: itUniones ← CrearIt(s.uniones)
8: while HaySiguiente?(itUniones) do
9:     itUnionesComercios ← CrearIt(comercios(Siguiente(itUniones)))
10:    while HaySiguiente?(itUnionesComercios) do
11:        tmpComercios ← comercios(Siguiente(itUnionesComercios))
12:        itComerciosUnion ← CrearIt(tmpComercios)
13:        while HaySiguiente?(itComerciosUnion) do
14:            comercioUnion ← Siguiente(itComerciosUnion)
15:            esta? ← false
16:            itComerciosRes ← CrearIt(comercios)
17:            while HaySiguiente?(itComerciosRes) do
18:                comercioRes ← Siguiente(itComerciosRes)
19:                if (comercioUnion.casilla = comercioRes.casilla) then
20:                    esta? ← true
21:                    if (comercioUnion.antigedad > comercioRes.antigedad) then
22:                        comercioRes.antigedad ← comercioUnion.antigedad
23:                    end if
24:                end if
25:                Avanzar(itComercios)
26:            end while
27:            if (¬esta?) then
28:                agregarRapido(comercios, comercioUnion)
29:            end if
30:            Avanzar(itComercios)
31:        end while
32:        agregar(comercios, tmpComercios)
33:        Avanzar(itUnionesComercios)
34:    end while
35:    Avanzar(itUniones)
36: end while
37: return comercios
```

Complejidad: Es el costo de agarrar las comercios del simCity actual, con recursion agarrar las comercios de máxima antigüedad por cada casillero de sus uniones e iterar sobre ellos

iNivel(**in** s : `estr`, **in** c : `Casilla`) → res : nat

```
1: itCasas ← CrearIt(casas(s))
2: while HaySiguiente?(itCasas) do
3:     casa ← Siguiente(itCasas)
4:     if casa.casilla = c then
5:         return casa.antiguedad
6:     end if
7:     Avanzar(itCasas)
8: end while
9: itComercios ← CrearIt(comercios(s))
10: while HaySiguiente?(itComercios) do
11:     comercio ← Siguiente(itComercios)
12:     if comercio.casilla = c then
13:         itCasas ← CrearIt(casas(s))
14:         nivelRes ← comercio.antiguedad
15:         while HaySiguiente?(itCasas) do
16:             casa ← Siguiente(itCasas)
17:             if distancia(casa.casilla, comercio.casilla) < 3 ∧ casa.antiguedad > nivelRes then
18:                 nivelRes ← casa.antiguedad
19:             end if
20:             Avanzar(itUniones)
21:         end while
22:         return nivelRes
23:     end if
24:     Avanzar(itUniones)
25: end while
```

Complejidad: O(#(comercios(s)) + #(casas(s))). El peor caso es cuando recorremos toda la lista de comercios. Además, la lista de casas se recorre por completo siempre.

---

iDistancia(**in** $c1$: `casilla`, **in** $c2$: `casilla`) → res : nat

```
1: return |π1(c1) - π1(c2)| + |π2(c1) - π2(c2)|
```

Complejidad: O(1)

---

iUnir(**in/out** $s1$: `estr`, **in** $s2$: `estr`)

```
1: agregarRapido(s1.uniones, s2)
2: s1.huboConstruccion = s1.huboConstruccion ∨ s2.huboConstruccion
3: s1.popularidad ← s1.popularidad + s2.popularidad + 1
4: if s1.turnoActual < s2.turnoActual then
5:     s1.turnoActual ← s2.turnoActual
6: end if
```

Complejidad: O(1)