DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico de Diseño

Algoritmos y Estructuras de Datos II - Grupo 36

Integrante	LU	Correo electrónico
Sebastián Silvera	680/17	sebaok2011@gmail.com
Luciano Zinik	290/17	lzinik@gmail.com
Martín Funes	342/16	martinfunesfunes@gmail.com
Fernando Regert	282/15	fernandostds9@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. 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.
Aliasing: los elementos hs y vs se agregan por copia.

HAYRIO(**in** m: `mapa`,**in** c: <`nat, nat`>) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} hayRio(m, c)\}$
Complejidad: $O(1)$
Descripción: Valida si en la casilla pasada hay un río.

AGREGARRIO(**in/out** m: `mapa`,**in** d: `direccion`, **in** p: `posicion`)
Pre $\equiv$ $\{m = m_0\}$
Post $\equiv$ $\{res =_{\text{obs}} agregarRio(m_0, d, p)\}$
Complejidad: $O(1)$
Descripción: Agrega un río al mapa.

UNIRMAPA(**in** m_1 : `mapa`,**in** m_2 : `mapa`) $\rightarrow$ res : `mapa`
Pre $\equiv$ $\{\text{vacio?}(m_1 \cap m_2)\}$
Post $\equiv$ $\{res =_{\text{obs}} unirMapa(m_1, m_2)\}$
Complejidad: $O(copy(m_1.horizontales \cup m_2.horizontales) + copy(m_1.verticales \cup m_1.verticales))$
Descripción: Une dos mapas.
Aliasing: los elementos m_1 y m_2 se agregan 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.

`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) $\equiv$ agregarRios(nuevoMapa, estr.verticales, estr.horizontales)

Algoritmos

---

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

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

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

---

iAgregarRio(**in/out** m : `estr`, **in** d : `Direccion`, **in** p : `Posicion`)

1: **if** $d = vertical$ **then**
2: $\quad m.verticales \leftarrow Ag(m.verticales, p)$
3: **else**
4: $\quad m.horizontales \leftarrow Ag(m.horizontales, p)$
5: **end if**

Complejidad: $O(copy(Ag(m.verticales, p)) + copy(Ag(m.horizontales, p)))$

---

iUnirMapa(**in** m_1 : `estr`, **in** m_2 : `estr`) $\rightarrow$ res : estr

1: $res.horizontales \leftarrow m_1.horizontales$
2: $it \leftarrow CrearIt(m_2.horizontales)$
3: **while** $haySiguiente(it)$ **do**
4: $\quad Agregar(res, siguiente(m_2.horizontales))$
5: **end while**
6: $res.verticales \leftarrow m_1.verticales$
7: $it \leftarrow CrearIt(m_2.verticales)$
8: **while** $haySiguiente(it)$ **do**
9: $\quad Agregar(res, Siguiente(m_2.horizontales))$
10: **end while**

Complejidad: $O(copy(m_1.horizontales \cup m_2.horizontales) + copy(m_1.verticales \cup m_2.verticales))$

---

iHayRio(**in/out** m : `estr`, **in** c : <`nat, nat`>) $\rightarrow$ res : bool

1: **if** $pertenece?(m.horizontales, c.prim)$ && $pertenece?(m.verticales, c.seg)$ **then**
2: $\quad res \leftarrow true$
3: **else**
4: $\quad res \leftarrow false$
5: **end if**

Complejidad: $O(pertenece?(m.horizontales, c.prim) + pertenece?(m.verticales, c.seg))$

---

2. Módulo SimCity

Interfaz

se explica con: SimCity

géneros: `simCity`

Operaciones básicas de mapa

nuevoJuego(**in** m: `mapa`) $\rightarrow$ res : `simCity`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} empezarPartida(m)\}$
Complejidad: $O(1)$
Descripción: crea un nuevo juego

agCasa(**in/out** s: `simCity`, **in** c: `Casilla`)
Pre $\equiv$ $\{s = s_0 \wedge sePuedeConstruir(s, c)\}$
Post $\equiv$ $\{res =_{\text{obs}} agregarCasa(s_0, c)\}$
Complejidad: $O(1)$
Descripción: en caso de ser posible, agrega una casa al juego dado.
Aliasing: el elemento c se agrega por copia.

agComercio(**in/out** s: `simCity`, **in** c: `Casilla`)
Pre $\equiv$ $\{s = s_0 \wedge sePuedeConstruir(s, c)\}$
Post $\equiv$ $\{res =_{\text{obs}} agregarComercio(s_0, c)\}$
Complejidad: $O(1)$
Descripción: en caso de ser posible, agrega un comercio al juego dado.
Aliasing: el elemento c se agrega por copia.

avanzarTurno(**in/out** s: `simCity`)
Pre $\equiv$ $\{s = s_0 \wedge huboConstruccion(s_0)\}$
Post $\equiv$ $\{s = avanzarTurno(s_0)\}$
Complejidad: $O(max(listaCasas, listaComercios))$
Descripción: avanzamos un turno en el juego.

unir(**in** s_1 : `simCity`, **in** s_2 : `simCity`) $\rightarrow$ res : `simCity`
Pre $\equiv$ $\{(\forall\, c : Casilla)(c \in construcciones(s_1) \Rightarrow \neg hayRio(mapa(s_2, c)) \wedge_L c \in construcciones(s_2) \Rightarrow$
$\neg hayRio(mapa(s_1, c))) \wedge_L (\forall\, c_1, c_2 : Casilla)(c_1 \in construcciones(s_1) \wedge\ c_2 \in construcciones(s_2) \Rightarrow$
$(esCasillaDeMaximoNivel(s_1, c_1) \wedge esCasillaDeMaximoNivel(s_2, c_2) \Rightarrow c_1 \neq c2_2))\}$
Post $\equiv$ $\{res = unir(s_1, s_2)\}$
Complejidad: $O(copy(listaComercios))$
Descripción: unir dor mapas.
Aliasing: los elementos s_1 y s_2 son pasados por referencia.

verMapa(**in** s: `simCity`) $\rightarrow$ res : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} mapa(s)\}$
Complejidad: $O(copy(mapa))$
Descripción: devuelve el mapa del juego dado.

verCasas(**in** s: `simCity`) $\rightarrow$ res : `Conj(Casilla)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} casas(s)\}$
Complejidad: $O(n^2)$
Descripción: devuelve las posiciones de las casas del juego dado.

verComercio(**in** s: `simCity`) $\rightarrow$ res : `Conj(Casilla)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} comercios(s)\}$
Complejidad: $O(n^2)$

Descripción: devuelve las posiciones de los comercios del juego dado.

VERNIVEL(**in** s : `simCity`, **in** c : `Casilla`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {hayConstruccion(s, c)}
Post $\equiv$ $\{res =_{\text{obs}} nivel(s, c)\}$
Complejidad: $O(n^2)$
Descripción: devuelve el nivel de la construccion que se encuentra en la casilla enviada.

HUBOCONSTRUCCION?(**in** s : `simCity`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} huboConstruccion(s)\}$
Complejidad: $O(1)$
Descripción: devuelve true si al menos una construccion en el juego enviado.

VERPOPULARIDAD(**in** s : `simCity`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} popularidad(s)\}$
Complejidad: $O(1)$
Descripción: devuelve la popularidad del juego dado.

ANTIGUEDAD(**in** s : `simCity`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} antiguedad(s)\}$
Complejidad: $O(1)$
Descripción: devuelve la antiguedad del juego dado.

Representación

Representación de SimCity

`simCity` se representa con `estr`

donde `estr` es `tupla`(*listaCasas*: `listaEnlazada(casilla, nivel)`,
listaComercios: `listaEnlazada(casilla, nivel)`,
construccionesNuevas: `bool`,
turno: `nat`,
map: `mapa`,
popularidad: `nat`)

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\iff$ $(\forall c : casilla)(\forall i, j : nat)(i \neq j \wedge_L < c, i > \in e.casas \Rightarrow_L < c, j > \in e.casas) \wedge$
$(\forall c : casilla)(\forall i, j : nat)(i \neq j \wedge_L < c, i > \in e.comercios \Rightarrow_L < c, j > \in e.comercios) \wedge$
$(\forall c : casilla)(\forall i, j : nat)(< c, i > \in e.casas \Rightarrow_L < c, j > \in e.comercios) \wedge$
$(\forall c : casilla)(\forall i, j : nat)(< c, i > \in e.comercios \Rightarrow_L < c, j > \in e.casas) \wedge$
$(\forall c : casilla)(\forall i : nat)(< c, i > \in e.casas \Rightarrow_L \neg hayRio(c, e.mapa)) \wedge$
$(\forall c : casilla)(\forall i : nat)(< c, i > \in e.comercios \Rightarrow_L \neg hayRio(c, e.mapa)) \wedge$
$(\forall c : casilla)(\forall i : nat)(< c, i > \in e.casas \Rightarrow_L i \Leftarrow e.turno) \wedge$
$(\forall c : casilla)(\forall i : nat)(< c, i > \in e.comercios \Rightarrow_L i \Leftarrow e.turno) \wedge$
$(\forall c : casilla)(\exists i : nat)(< c, i > \in e.casas \vee_L < c, i > \in e.comercios \Rightarrow_L i = e.turno) \wedge$
$(\forall c, d, f : casilla)(\forall i, j, k : nat)(< c, i > \in e.comercios \wedge_L < d, j >, < f, k > \in e.casas \wedge_L (i, j \Leftarrow k) \wedge_L d, f \in$
$casasADistancia3Manhattan(e.casas, c) \Rightarrow_L i = k)$

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

Abs(e) $\equiv$ $(\forall\, e : estr)Abs(e) =_{\text{obs}} s : simCity/$
$longitud(e.casas) = \#casas(s) \wedge$
$(\forall < c, n >:< casilla, nivel >)(< c, n > \in e.casas \Rightarrow (\exists c_0 : casilla)(\Pi_1(< c, n >) = c_0 \wedge c_0 \in casas(s) \wedge \Pi_2(< c, n >) = nivel(s, c_0)))$
$\wedge$
$(\forall < c, n >:< casilla, nivel >)(< c, n > \in e.comercios \Rightarrow (\exists c_0 : casilla)(\Pi_1(< c, n >) = c_0 \wedge c_0 \in comercios(s)\Pi_2(< c, n >) = nivel(s, c_0)))$
$\wedge$
$e.popularidad = popularidad(s)$
$\wedge$
$mapa(s) = e.map$
$\wedge$
$huboConstruccion(s) = e.construccionesNuevas$
$\wedge$
$(\forall\, c : casilla)(c \in casas(s) \vee c \in comercios(s) \Rightarrow (\exists\, j : nat)(nivel(s, c) \Leftarrow j \wedge (\exists\, c_0 : casilla)(c_0 \in casas(s) \vee c_0 \in comercios(s)) \wedge nivel(s, c_0) = j \wedge e.turno = j))$

Algoritmos

---

iNuevoJuego(**in** m : `mapa`) $\rightarrow$ res : simCity

1: $res.casas \leftarrow Vacia()$
2: $res.comercios \leftarrow Vacia()$
3: $res.turno \leftarrow 0$
4: $res.mapa \leftarrow m$
5: $res.popularidad \leftarrow 0$
6: **return** res

Complejidad: $O(1)$

---

iAgCasa(**in/out** s : `simCity`, **in** c : `casilla`)

1: $AgregarAtras(s.casas, (c, 0))$
2: $s.construccionesNuevas \leftarrow true$

Complejidad: $O(1)$

---

iAgComercio(**in/out** s : `simCity`, **in** c : `casilla`)

1: $AgregarAtras(s.comercio, (c, 0))$
2: $s.construccionesNuevas \leftarrow true$

Complejidad: $O(1)$

---

iAvanzarTurno(**in/out** s : `simCity`)

1: $s.turno++$
2: $s.construccionesNuevas \leftarrow false$
3:
4: $itCa \leftarrow crearIT(s.listaCasas)$
5: **while** $haySiguiente(itCa)$ **do**
6: $\quad siguiente(itCa).nivel++$
7: $\quad Avanzar(itCa)$
8: **end while**
9:
10: $itCo \leftarrow crearIT(s.listaComercios)$
11: **while** $haySiguiente(itCo)$ **do**
12: $\quad siguiente(itCo).nivel++$
13: $\quad Avanzar(itCo)$
14: **end while**

Complejidad: $O(max(listaCasas, listaComercios))$
Justificacion: Avanzar turno actualiza de forma simple las ciertas variables, que cambian al cambiar el turno, como el nivel de las construcciones, el turno y el bool que representa la existencia o no de construcciones hechas en el turno presente.

---

iUnir(**in/out** s_1 : `simCity`, **in** s_2 : `simCity`) $\rightarrow$ res : simCity

1: $res.listaCasas \leftarrow s_1.listaCasas$
2: $it \leftarrow crearIT(s_2.listaCasas)$
3: **while** haySiguiente(it) **do**
4: $\quad res.listaCasas.agregarSiguiente(it)$
5: **end while**
6: $res.listaComercios \leftarrow s_1.listaComercios$
7: $it \leftarrow crearIT(s_2.listaComercios)$
8: **while** $haySiguiente(it)$ **do**
9: $\quad res.listaComercios.AgregarSiguiente(it)$
10: **end while**
11: $res.construccionesNuevas \leftarrow false$
12: $res.turno \leftarrow max(s_1.turno, s_2.turno)$
13: $res.mapa \leftarrow unirMapas(s_1.mapa, s_2.mapa)$
14: $res.popularidad \leftarrow s_1.popularidad + s_2.popularidad + 1$

Complejidad: $O(copy(listaComercios))$
Justificacion: Aquí obtenemos el juego resultante de la unión. Se copian las estructuras que representan a las construcciones de forma que se unen las de cada juego. Lo mismo sucede con el mapa.

---

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

1: $res \leftarrow s.mapa$

Complejidad: $O(copy(mapa))$

---

iVerCasas(**in** s : `simCity`) $\rightarrow$ res : Conj(casilla)

```
1: arreglarConstrucciones(s)
2: res ← Vacio()
3: it ← crearIt(s.listaCasas)
4: while HaySiguiente(it) do
5:     AgregarAtras(res, siguiente(it))
6:     Avanzar(it)
7: end while
```

Complejidad: $O(arreglarConstrucciones(s))$

---

iVerComercios(**in** s : `simCity`) $\rightarrow$ res : Conj(casilla)

```
1: arreglarConstrucciones(s)
2: res ← Vacio()
3: it ← crearIt(s.listaComercios)
4: while HaySiguiente(it) do
5:     AgregarAtras(res, siguiente(it))
6:     Avanzar(it)
7: end while
```

Complejidad: $O(arreglarConstrucciones(s))$

---

iVerNivel(**in** s : `simCity`, **in** c : `Casilla`) $\rightarrow$ res : Nat

```
1: arreglarConstrucciones(s)
2: construccion ← devolverConstruccion(c)
3: if pertenece(construccion, listaComercios) then
4:     for casainlistaCasas do
5:         if distanciaMenorA4(construccion.casilla, casa.casilla) then
6:             construccion.nivel ← max(construccion.nivel, casa.nivel)
7:         end if
8:     end for
9: end if
10: res ← construccion.nivel
11: return res
```

Complejidad: $O(arreglarConstrucciones(s))$

---

iHuboConstruccion?(**in** s : `simCity`) $\rightarrow$ res : bool

```
1: res ← s.construccionesNuevas
```

Complejidad: $O(1)$

---

iVerPopularidad(**in** s : `simCity`) $\rightarrow$ res : nat

```
1: res ← s.popularidad
```

Complejidad: $O(1)$

---

iVerAntiguedad(**in** s : `simCity`) $\rightarrow$ res : nat

```
1: res ← s.turno
```

Complejidad: $O(1)$

---

devolverConstruccion(**in** s : `simCity`, **in** c : `casilla`) $\rightarrow$ res : $< casilla, nivel >$

```
1: for casa in s.listaCasas do
2:     if c = casa.casilla then
3:         res ←< c, casa.nivel >
4:     end if
5: end for
6: for comercio in s.listaComercios do
7:     if c = comercio.casilla then
8:         res ←< c, comercio.nivel >
9:     end if
10: end for
```

Complejidad: $O(n)$

---

distanciaMenorA4(**in** $< x_1, y_1 >$: `<nat, nat>`, **in** $< x_2, y_2$: `<nat, nat>`) $\rightarrow$ res : bool

```
1: res ← (abs(x_1 − x_2) + abs(y_1 − y_2)) < 4
```

Complejidad: $O(1)$

arreglarConstrucciones(**in/out** s : `simCity`)

```
1: itCa ← crearIT(s.listaCasas)
2: itCa2 ← crearIT(s.listaCasas)
3: while haySiguiente(itCa) do
4:     while haySiguiente(itCa2) do
5:         if siguiente(itCa).casilla = siguiente(itCa2).casilla then
6:             if siguiente(itCa).nivel > siguiente(itCa2).nivel then
7:                 eliminarSiguiente(itCa2)
8:             else
9:                 eliminarSiguiente(itCa2)
10:            end if
11:        end if
12:        Avanzar(itCa2)
13:    end while
14:    Avanzar(itCa)
15: end while
16:
17: itCo ← crearIT(s.listaComercios)
18: itCo2 ← crearIT(s.listaComercios)
19: while haySiguiente(itCo) do
20:    while haySiguiente(itCo2) do
21:        if siguiente(itCo).casilla = siguiente(itCo2).casilla then
22:            if siguiente(itCo).nivel > siguiente(itCo2).nivel then
23:                eliminarSiguiente(itCo2)
24:            else
25:                eliminarSiguiente(itCo)
26:            end if
27:        end if
28:        Avanzar(itCo2)
29:    end while
30:    Avanzar(itCo)
31: end while
32:
33: itCa ← crearIT(s.listaCasas)
34: itCo ← crearIT(s.listaComercios)
35: while haySiguiente(itCa) do
36:    while haySiguiente(itCo) do
37:        if haySiguiente(itCa.casilla = siguiente(itCo).casilla then
38:            if siguiente(itCa).nivel > siguiente(itCo).nivel then
39:                eliminarSiguiente(itCo)
40:            else
41:                eliminarSiguiente(itCa)
42:            end if
43:        end if
44:        Avanzar(itCo)
45:    end while
46:    Avanzar(itCa)
47: end while
```

Complejidad: $O(max(listaCasas, listaComercios)^2)$

Justificacion: arreglarConstrucciones es la función auxiliar encargada de solucionar las decisiones que se deben resolver en el tad al unir dos juegos. Se encarga de borrar construcciones repetidas en las listas de construcciones.