DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Número 2

Algoritmos y Estructuras de Datos II

Grupo: 21

Integrante	LU	Correo electrónico
Langberg, Andrés	249/14	andreslangberg@gmail.com
Walter, Nicolás	272/14	nicowalter25@gmail.com
Sticco, Patricio Bernardo	337/14	pbsticco@hotmail.com
Len, Julián	467/14	julianlen@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. Diseño del Tipo Campus

1.1. Especificación

Se usa el Tad Campus especificado por la cátedra.

1.2. Aspectos de la interfaz

1.2.1. Interfaz

Se explica con especificación de Campus

Género `campus`

Operaciones básicas de Campus

CrearCampus(**in** c: nat, **in** f: nat) $\longrightarrow res : campus$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} crearCampus(c, f)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: Crea un campus de c columnas y f filas.

Filas?(**in** c: $campus$) $\longrightarrow res : nat$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} filas(c)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve la cantidad de filas en el campus.

Columnas?(**in** c: $campus$) $\longrightarrow res : nat$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} columnas(c)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve la cantidad de columnas en el campus.

Ocupada?(**in** c: $campus$, **in** p: $posicion$) $\longrightarrow res : bool$
Pre $\equiv$ { $posValida(p, c)$ }
Post $\equiv$ { $res =_{\text{obs}} ocupada?(p, c)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve $true$ sii p esta ocupada por un obstaculo.

AgregarObstaculo(**in/out** c: $campus$, **in** p: $posicion$) $\longrightarrow$
Pre $\equiv$ { $c =_{\text{obs}} c_0 \wedge$ posValida(p,c) $\wedge_{\text{L}}$ $\neg$ocupada?(p,c)}
Post $\equiv$ { $c =_{\text{obs}} agregarObstaculo(p, c_0)$}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve $true$ sii p esta ocupada por un obstaculo.

PosValida?(**in** c: $campus$, **in** p: $posicion$) $\longrightarrow res : bool$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} posValida?(p, c)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve $true$ sii p es parte del mapa.

EsIngreso?(**in** c: $campus$, **in** p: $posicion$) $\longrightarrow res : bool$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} esIngreso?(p, c)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve $true$ sii p es un ingreso.

VECINOS(**in** c: $campus$, **in** p: $posicion$) $\longrightarrow res : conj(posicion)$
Pre $\equiv$ { $posValida(p, c)$ }
Post $\equiv$ { $res =_{\text{obs}} vecinos(p, c)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve el conjunto de posiciones vecinas a p.

PROXPOSICION(**in** c: $campus$, **in** dir: $direccion$, **in** p: $posicion$) $\longrightarrow res : posicion$
Pre $\equiv$ { $posValida(p, c)$ }
Post $\equiv$ { $res =_{\text{obs}} proxPosicion(p, d, c)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve la posicion vecina a p que esta en la direccion dir.

INGRESOSMASCERCANOS(**in** c: $campus$, **in** p: $posicion$) $\longrightarrow res : conj(posicion)$
Pre $\equiv$ { $posValida(p, c)$ }
Post $\equiv$ { $res =_{\text{obs}} ingresosMasCercanos(p, c)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve el conjunto de ingresos mas cercanos a p.

1.3. Pautas de implementación

1.3.1. Estructura de representación

campus **se representa con** *estr*
donde *estr* **es**
tupla(
filas: ***nat*** $\times$
columnas: ***nat*** $\times$
mapa: ***vector(vector(bool))***
)

1.3.2. Justificación

1.3.3. Invariante de Representación

Informal

1. El mapa debe tener tantas filas como indica la estructura, lo mismo con las columnas.

Formal

Rep : estr $\longrightarrow$ *boolean*
($\forall$ e : estr)
Rep(e) $\equiv$ ($true \iff$
(1) e.filas = longitud(e.mapa) $\wedge_{\text{L}}$ ($\forall$ i : nat)(i $\leq$ e.filas $\Rightarrow$ longitud(e.mapa[i])= e.columnas))

1.3.4. Función de Abstracción

Abs : estr e $\longrightarrow$ campus $\{\text{Rep}(e)\}$
($\forall$ e:estr) Abs(e) $=_{\text{obs}}$ c : campus /
$\Big($filas(c) = e.filas $\wedge$ columnas(c) = e.columnas $\wedge_{\text{L}}$ ($\forall$ p : posicion)(p.X $\leq$ e.filas $\wedge$
p.Y $\leq$ e.columnas $\Rightarrow_{\text{L}}$ ocupada?(p,c) $\Leftrightarrow$ (e.mapa[f])[c]$\Big)$

1.3.5. Algoritmos

```
1: function iCrearCampus(in c: nat, in f: nat)⟶ res : estr                ▷ O(f^2 * c^2)
2:     var vector(vector(bool)) mapa ← vacia(vacia())                      ▷ O(1)
3:     var nat i ← 0                                                       ▷ O(1)
4:     while i≤f do                                                        ▷ O(f)
5:         var vector(bool) nuevo ← vacia()                                ▷ O(1)
6:         var nat j ← 0                                                   ▷ O(1)
7:         while j≤c do                                                    ▷ O(c)
8:             AgregarAtras(nuevo, false)                                  ▷ O(c)
9:             j++                                                         ▷ O(1)
10:        end while
11:        AgregarAtras(mapa, nuevo)                                       ▷ O(f)
12:        i++                                                             ▷ O(1)
13:    end while
14:    res ← < f, c, mapa>                                                 ▷ O(1)
15: end function
```

```
1: function iAgregarObstaculo(in/out e: estr, in p: posicion)⟶ res : estr   ▷ O(longitud(e.mapa[p.X])
2:     Agregar(e.mapa[p.X], p.Y, true)                                      ▷ O(longitud(e.mapa[p.X]))
3: end function
```

```
1: function iFilas?(in e: estr)⟶ res : nat                                 ▷ O(1)
2:     res ← e.filas                                                        ▷ O(1)
3: end function
```

```
1: function iColumnas?(in e: estr) ⟶ res : nat                                  ▷ O(1)
2:     res ← e.columnas                                                         ▷ O(1)
3: end function
```

```
1: function iOcupada?(in e: estr, in p: posicion) ⟶ res : bool                  ▷ O(1)
2:     res ← (e.mapa[p.X])[p.Y]                                                 ▷ O(1)
3: end function
```

```
1: function iPosValida?(in e: estr, in p: posicion) ⟶ res : bool                ▷ O(1)
2:     res ← (0 < p.X) ∧ (p.X ≤ e.filas) ∧ (0 < p.Y) ∧ (p.Y ≤ e.columnas)       ▷ O(1)
3: end function
```

```
1: function iEsIngreso?(in e: estr, in p: posicion) ⟶ res : bool                ▷ O(1)
2:     res ← (p.Y = 1) ∨ (p.Y = e.filas)                                        ▷ O(1)
3: end function
```

```
1: function iVecinos(in e: estr, in p: posicion) ⟶ res : conj(posicion)         ▷ O(1)
2:     var conj(posicion) nuevo ← vacio()                                       ▷ O(1)
3:     Agregar(nuevo, (p.X+1,p.Y))
4:     Agregar(nuevo, (p.X-1,p.Y))
5:     Agregar(nuevo, (p.X,p.Y+1))
6:     Agregar(nuevo, (p.X,p.Y-1))
7:     var itConj(posicion) it ← crearIt(nuevo)
8:     while haySiguiente(it) do                                                ▷ O(c)
9:         if iPosValida?(e,siguiente(it)) then                                 ▷ O(1)
10:            avanzar(it)                                                      ▷ O(1)
11:        else
12:            eliminarSiguiente(it)                                            ▷ O(1)
13:        end if
14:    end while
15:    res ← nuevo                                                              ▷ O(1)
16: end function
```

```
1: function iVecinosComunes(in e: estr, in p: posicion, in p2: posicion) ⟶ res : conj(posicion)   ▷ O()
2:     var conj(posicion) v ← vecinos(e,p)
3:     var conj(posicion) v2 ← vecinos(e,p2)
4:     var conj(posicion) nuevo ← vacio()
5:     var itConj(posicion) it ← crearIt(v)
6:     while haySiguiente(it) do                                                ▷ O(c)
7:         if pertenece?(v2,siguiente(it)) then                                 ▷ O(1)
8:             Agregar(nuevo, siguiente(it))
9:         end if
10:        avanzar(it)                                                          ▷ O(1)
11:    end while
12:    res ← nuevo                                                              ▷ O(1)
13: end function
```