1. Módulo Campus

Interfaz

se explica con: CAMPUS

géneros: `campus`

Operaciones básicas de campus

NUEVOCAMPUS(**in** al : `nat`, **in** an : `nat`) $\rightarrow$ res : `campus`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ crearCampus(al, an)}
Complejidad: $\Theta(1)$
Descripción: Crea un nuevo campus vacio de alto al x ancho an.

AGREGAROBSTACULO(**in** p : `pos`), **in/out** c : `campus`)
Pre $\equiv$ {posValida(p, c) $\wedge$ $\neg$ocupada?(p, c) $\wedge$ $c =_{\text{obs}} c_0$ }
Post $\equiv$ {$c =_{\text{obs}}$ agregarObstaculo(p, c_0)}
Descripción: Agrega un obstaculo al campus c en la posicion p.

FILAS(**in** c : `campus`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ filas(c}
Descripción: Devuelve el alto (filas) del campus c.

COLUMNAS(**in** c : `campus`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ columnas(c}
Descripción: Devuelve el ancho (columnas) del campus c.

OCUPADA?(**in** p : `pos`, **in** c : `campus`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ posValida?(p, c)}
Descripción: Devuelve $true$ si la posicion p es valida en el campus c, sino retorna $false$.

ESINGRESO?(**in** p : `pos`, **in** c : `campus`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ esIngreso?(p, c)}
Descripción: Verifica si la posicion p es una entrada del campus c.

INGRESOSUPERIOR?(**in** p : `pos`, **in** c : `campus`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ ingresoSuperior?(p, c)}
Descripción: Verifica si la posicion p es una entrada superior del campus c.

INGRESOINFERIOR?(**in** p : `pos`, **in** c : `campus`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ ingresoInferior?(p, c)}
Descripción: Verifica si la posicion p es una entrada inferior del campus c.

VECINOS(**in** p : `pos`, **in** c : `campus`) $\rightarrow$ res : `conj(`pos`)`
Pre $\equiv$ {posValida(p, c)}
Post $\equiv$ {$res =_{\text{obs}}$ vecinos(p, c)}
Descripción: Devuelve un conjunto de las posiciones que rodean a p en el campus c

DISTANCIA(**in** p_0 : `pos`, **in** p_1 : `pos`, **in** c : `campus`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ distancia(p_0, p_1, c)}
Descripción: Devuelve la distancia, en casilleros, desde la posicion p_0 a la posicion p_1.

ProxPosicion(**in** p: `pos`, **in** d: `dir`, **in** c: `campus`) $\rightarrow$ res : `pos`
Pre $\equiv$ {posValida(p, c)}
Post $\equiv$ {$res =_{\text{obs}}$ proxPosicion(p, d, c)}
Descripción: Indica la posicion que se encuentra al lado de p, en la direccion d.

Obstaculos(**in** c: `campus`) $\rightarrow$ res : `conj(`pos`)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ obstaculos(c)}
Descripción: Devuelve un conjunto que contiene todas las posiciones ocupadas por obstaculos en el campus c. EXTENDER TAD.

Representación

Representación de la lista

`campus` se representa con `estr`

donde `estr` es `tupla`(*alto*: `nat`, *ancho*: `nat`, *obstaculos*: `matriz(bool)`)
donde `pos` es `tupla`(*fila*: `nat`, *columna*: `nat`)

Invariante de representacion en castellano:

1. Para toda p de tipo *pos*, si p esta definida en obstaculos, entonces tanto la fila como la columna de p son menores o iguales a *alto* y *ancho* respectivamente.

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

1. ($\forall$ p : pos) $p \in$ claves(e.obstaculos) $\Rightarrow$ (p.fila $\leq$ c.alto $\wedge$ p.columna $\leq$ c.ancho)

Abs : estr e $\longrightarrow$ campus {Rep(e)}
Abs(e) $\equiv$ c : campus /
($\forall$ p : pos) def?(p, e.obstaculos) $=_{\text{obs}}$ ocupada?(p, c) $\wedge$ alto(c) $=_{\text{obs}}$ e.alto $\wedge$ ancho(c) $=_{\text{obs}}$ e.ancho

Algoritmos

Algoritmos de Campus

Lista de algoritmos

iNuevoCampus(**in** al : `nat`, **in** an : `nat`) → res: estr
begin
| res.computadoras ← vacio() //**O(1)**
| res.mapa ← vacia() //**O(1)**
| res.indexToString ← vacia() //**O(1)**
end
Complejidad: O(1)

Algoritmo 1: NuevoCampus

---

iAgregarObstaculo(**in** p : `pos`, **in/out** e : `estr`)
begin
| Colocar(p, $true$, e.obstaculos) //**O(1)**
end
Complejidad: O(1)

Algoritmo 2: AgregarObstaculo

---

iFilas(**in** e : `estr`) → res: nat
begin
| res ← e.alto //**O(1)**
end
Complejidad: O(1)

Algoritmo 3: Filas

---

iColumnas(**in** e : `estr`) → res: nat
begin
| res ← e.ancho //**O(1)**
end
Complejidad: O(1)

Algoritmo 4: Columnas

---

iOcupada?(**in** p : `pos`, **in** e : `estr`) → res: bool
begin
| Ocupada?(p, e.obstaculos) //**O(1)**
end
Complejidad: O(1)
Comentarios: La funcion Ocupada que se llama, es la del modulo matriz

Algoritmo 5: Ocupada?

---

*i*PosValida(**in** *p* : `pos`, **in** *e* : `estr`) → res: bool
begin
  res ← true //**O(1)**
  if *p.fila* >*e.alto* ∨ *p.columna* >*e.ancho* **then** //**O(1)**
    res ← false //**O(1)**
  end
  if *0* <*p.fila* ∨ *0* <*p.columna* **then** //**O(1)**
    res ← false //**O(1)**
  end
end
Complejidad: O(1)

Algoritmo 6: PosValida

*i*EsIngreso?(**in** *p* : `pos`, **in** *e* : `estr`) → res: bool
begin
  res ← false //**O(1)**
  if *ingresoSuperior(p, e)* ∨ *ingresoInferior(p, e)* **then** //**O(1)**
    res ← true //**O(1)**
  end
end
Complejidad: O(1)

Algoritmo 7: EsIngreso?

*i*IngresoSuperior?(**in** *p* : `pos`, **in** *e* : `estr`) → res: bool
begin
  res ← false //**O(1)**
  if *p.fila* == *1* **then** //**O(1)**
    res ← true
  end
end
Complejidad: O(1)

Algoritmo 8: IngresoSuperior?

*i*IngresoInferior?(**in** *p* : `pos`, **in** *e* : `estr`) → res: bool
begin
  res ← false //**O(1)**
  if *p.fila* == *e.alto* **then** //**O(1)**
    res ← true
  end
end
Complejidad: O(1)

Algoritmo 9: IngresoInferior?