1. Modulo HippiesYEstudiantes

Interfaz

se explica con: HippiesYEstudiantes

generos: `hipyest`

Operaciones basicas de Hippies y Estudiantes

NuevoHippiesYEstudiantes() $\rightarrow$ *res* : `hipyest`
Pre $\equiv$ {true}
Post $\equiv$ {*res* $=_{\text{obs}}$ nuevoHippiesYEstudiantes()}
Complejidad: $\Theta(1)$
Descripción: Crea un nuevo contenedor de Hippies y Estudiantes vacio

DefinirHippie(**in** *n*: `nombre`, **in** *p*: `posicion`, **in/out** *he*: `hipyest`)
Pre $\equiv$ {*he* $=_{\text{obs}}$ he_0}
Post $\equiv$ {*he* $=_{\text{obs}}$ defHippie(*n*, *p*, he_0)}
Complejidad: $O(n_m)$
Descripción: Agrega al hippie *n* a hippies y estudiantes *he*

DefinirEstudiante(**in** *n*: `nombre`, **in** *p*: `posicion`, **in/out** *he*: `hipyest`)
Pre $\equiv$ {*he* $=_{\text{obs}}$ he_0}
Post $\equiv$ {*he* $=_{\text{obs}}$ defEstudiante(*n*, *p*, he_0)}
Complejidad: $O(n_m)$
Descripción: Agrega al hippie *n* a hippies y estudiantes *he*

PosHippieYEstudiante(**in** *n*: `string`, **in** *he*: `hipyest`) $\rightarrow$ *res* : `posicion`
Pre $\equiv$ {esta?(*n*, *he*)}
Post $\equiv$ {*res* $=_{\text{obs}}$ posHippieYEstudiante(*n*, *he*)}
Complejidad: $O(n_m)$
Descripción: Devuelve la posicion del hippie o estudiante *n*.

Esta?(**in** *n*: `string`, **in** *he*: `hipyest`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {*res* $=_{\text{obs}}$ esta?(*n*, *he*)}
Complejidad: $O(n_m)$
Descripción: Devuelve *true* si *n* esta en hippies o estudiantes de *he*.

EsHippie?(**in** *n*: `string`, **in** *he*: `hipyest`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {esta?(*n*, *he*)}
Post $\equiv$ {*res* $=_{\text{obs}}$ esHip?(*n*, *he*)}
Complejidad: $O(n_m)$
Descripción: Devuelve *true* si *n* esta en hippies de *he*.

EsEst?(**in** *n*: `string`, **in** *he*: `hipyest`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {esta?(*n*, *he*)}
Post $\equiv$ {*res* $=_{\text{obs}}$ $\neg$esHip?(*n*, *he*)}
Complejidad: $O(n_m)$
Descripción: Devuelve *true* si *n* esta en estudiantes de *he*.

Estudiantes(**in** *he*: `hipyest`) $\rightarrow$ *res* : `conj(nombre)`
Pre $\equiv$ {true}
Post $\equiv$ {alias(*res* $=_{\text{obs}}$ estudiantes?(*he*))}
Complejidad: $O(1)$
Descripción: Devuelve el conjunto de estudiantes.
Aliasing: *res* no es modificable. Si se agregan hippies o estudiantes, *res* queda invalidado.

Hippies(**in** *he*: `hipyest`) $\rightarrow$ *res* : `conj(nombre)`
Pre $\equiv$ {true}

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{hippies?}(he))\}$
Complejidad: O(1)
Descripción: Devuelve el conjunto de hippies.
Aliasing: *res* no es modificable. Si se agregan hippies o estudiantes, *res* queda invalidado.

BORRAR(**in** n : `nombre`, **in/out** he : `hipyest`)
Pre $\equiv \{he =_{\text{obs}} he_0\}$
Post $\equiv \{he =_{\text{obs}} \text{borrar}(n, p, he_0)\}$
Complejidad: $O(n_m)$
Descripción: Agrega al hippie n a hippies y estudiantes he

Representación

Representacion de HippiesYEstudiantes

`hipyest)` **se representa con** `estr`

donde `estr` es `tupla`(*estudiantes*: `DiccClavesRapidas(nombre, posicion)` , *hippies*: `DiccClavesRapidas(nombre, posicion)`)

Invariante de representacion en castellano:

1. No hay ningun nombre en *estudiantes* que tambien este en *hippies*

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

1. ($\forall$ n : nombre)
   (Def?(n, e.estudiantes) $\Rightarrow$ $\neg$Def?(n, e.hippies)) $\wedge$
   (Def?(n, e.hippies) $\Rightarrow$ $\neg$Def?(n, e.estudiantes))

Abs : estr e $\longrightarrow$ HippiesYEstudiantes {Rep(e)}
Abs(e) $\equiv$ he : HippiesYEstudiantes /
($\forall$ n : nombre)
(Def?(n, e.estudiantes) $\iff$ (def?(n, diccPos(he)) $\wedge_{\text{L}}$ $\neg$esHip(n, he))) $\wedge$
(Def?(n, e.hippies) $\iff$ (def?(n, diccPos(he)) $\wedge_{\text{L}}$ esHip(n, he))) $\wedge_{\text{L}}$
(Def?(n, e.estudiantes) $\Rightarrow_{\text{L}}$ Obtener(n, e.estudiantes) $=_{\text{obs}}$ obtener(n, diccPos(he))) $\wedge$
(Def?(n, e.hippies) $\Rightarrow_{\text{L}}$ Obtener(n, e.hippies) $=_{\text{obs}}$ obtener(n, diccPos(he)))

Algoritmos

Algoritmos de Agentes

Lista de algoritmos

1. NuevoHippieYEst . . . 3
2. PosHippieYEstudiante . . . 3
3. DefinirHippie . . . 3
4. DefinirEst . . . 3
5. Esta? . . . 3
6. EsEst? . . . 3
7. EsHip? . . . 4
8. Estudiantes . . . 4
9. Hippies . . . 4
10. Borrar . . . 4

*i*NuevoHippieYEst() → res: estr
begin
| res.hippies ← Vacio() //**O(1)**
| res.estudiantes ← Vacio() //**O(1)**
end
Complejidad: O(1)

Algoritmo 1: NuevoHippieYEst

---

*i*PosHippieYEstudiante(**in** n : `nombre`, **in** he : `estr`) → res: posicion
begin
| **if** *EsEst?*(n, he) **then** //**O**(n_m)
| | res ← Obtener(n, he.estudiantes) //**O**(n_m)
| **else**
| | res ← Obtener(n, he.hippies) //**O**(n_m)
| **end**
end
Complejidad: O(n_m)

Algoritmo 2: PosHippieYEstudiante

---

*i*DefinirHippie(**in** n : `nombre`, **in** p : `posicion`, **in/out** he : `estr`)
begin
| Definir(n, p, he.hippies) //**O**(n_m)
end
Complejidad: O(n_m)

Algoritmo 3: DefinirHippie

---

*i*DefinirEst(**in** n : `nombre`, **in** p : `posicion`, **in/out** he : `estr`)
begin
| Definir(n, p, he.estudiantes) //**O**(n_m)
end
Complejidad: O(n_m)

Algoritmo 4: DefinirEst

---

*i*Esta?(**in** n : `nombre`, **in** he : `estr`) → res: bool **begin**
| res ← Def?(n, he.hippies) ∨ Def?(n, he.estudiantes) //**O**$(2n_m)$
end
Complejidad: O(n_m)

Algoritmo 5: Esta?

---

*i*EsEst?(**in** n : `nombre`, **in** he : `estr`) → res: bool
begin
| res ← Def?(n, he.estudiantes) //**O**(n_m)
end
Complejidad: O(n_m)

Algoritmo 6: EsEst?

---

*i*EsHip?(**in** n : `nombre`, **in** he : `estr`) → res: bool
begin
| res ← Def?(n, he.hippies) //**O**(n_m)
end
Complejidad: O(n_m)

Algoritmo 7: EsHip?

---

*i*Estudiantes(**in** he : `estr`) → res: puntero(conj(nombre))
begin
| res ← Claves(he.estudiantes) //**O(1)**
end
Complejidad: O(1)

Algoritmo 8: Estudiantes

---

*i*Hippies(**in** he : `estr`) → res: puntero(conj(nombre))
begin
| res ← Claves(he.hippies) //**O(1)**
end
Complejidad: O(1)

Algoritmo 9: Hippies

---

*i*Borrar(**in** n : `nombre`, **in/out** he : `estr`)
begin
| **if** *EsEst?*(n, he) **then** //**O**(n_m)
| | res ← Borrar(n, he.estudiantes) //**O**(n_m)
| **else**
| | res ← Borrar(n, he.hippies) //**O**(n_m)
| **end**
end
Complejidad: O(n_m)

Algoritmo 10: Borrar
