1. Módulo Mapa

1.1. Interfaz

se explica con: MAPA.
géneros: `map`.

1.1.1. Operaciones básicas de mapa

VACIO() $\rightarrow$ *res* : `map`
Pre $\equiv$ {true}
Post $\equiv$ {*res* $=_{\text{obs}}$ vacio()}
Complejidad: $O(1)$
Descripción: crea un mapa nuevo

AGREGAR(**in** *e* : `string`, **in/out** *m* : `map`)
Pre $\equiv$ {$e \notin$ estaciones(m) $\wedge$ $m =_{\text{obs}} m_0$}
Post $\equiv$ {$m =_{\text{obs}}$ agregar(e, m_0)}
Complejidad: $O(|e|)$
Descripción: agrega una estacion al mapa

CONECTAR(**in** *e*1 : `string`, **in** *e*2 : `string`, **in** *r* : `restriccion`, **in/out** *m* : `map`)
Pre $\equiv$ {$e1 \neq e2 \wedge$ ($e1 \in$ estaciones(m) $\wedge$ $e2 \in$ estaciones(m)) $\wedge_{\text{L}}$ ($\neg$ conectadas?($e1$, $e2$)) $\wedge$ $m =_{\text{obs}} m_0$}
Post $\equiv$ {$m =_{\text{obs}}$ conectar($e1$, $e2$, r, m_0)}
Complejidad: $O(|e1| + |e2|)$
Descripción: conecta dos estaciones previamente agregadas con su respectiva restriccion para la senda que forman

ESTA?(**in** *e* : `string`, **in** *m* : `map`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {*res* $=_{\text{obs}}$ $e \in$ estaciones(m)}
Complejidad: $O(|e|)$
Descripción: verifica si la estacion fue agregada a la ciudad

CONECTADAS?(**in** *e*1 : `string`, **in** *e*2 : `string`, **in** *m* : `map`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {($e1 \in$ estaciones(m) $\wedge$ $e2 \in$ estaciones(m)) $\wedge$ $m =_{\text{obs}} m_0$}
Post $\equiv$ {*res* $=_{\text{obs}}$ conectadas?($e1$, $e2$, m)}
Complejidad: $O(|e1| + |e2|)$
Descripción: Se fija si las dos estaciones estan conectadas segun el mapa

RESTRICCION(**in** *e*1 : `string`, **in** *e*2 : `string`, **in** *m* : `map`) $\rightarrow$ *res* : `restriccion`
Pre $\equiv$ {($e1 \in$ estaciones(m) $\wedge$ $e2 \in$ estaciones(m)) $\wedge_{\text{L}}$ (conectadas?($e1$, $e2$))}
Post $\equiv$ {*res* $=_{\text{obs}}$ restriccion($e1$, $e2$, m)}
Complejidad: $O(|e1| + |e2|)$
Descripción: devuelve la restriccion correspondiente a la senda que conecta las dos estaciones en el mapa

ESTACIONES(**in** *m* : `map`) $\rightarrow$ *res* : `itLista(string)`
Pre $\equiv$ {true}
Post $\equiv$ {SecuSuby(res) $=_{\text{obs}}$ estaciones(m)}
Complejidad: $O(1)$
Descripción: devuelve un iterador para las estaciones del mapa

SENDAS(**in** *m* : `map`) $\rightarrow$ *res* : `itLista(tupla(string, string))`
Pre $\equiv$ {true}
Post $\equiv$ {esPermutacion?(res, sendas(m))}
Complejidad: $O(1)$
Descripción: devuelve un iterador para los pares de estaciones que forman una senda

1.1.2. Operaciones Auxiliares del TAD

sendas : mapa m $\longrightarrow$ lista(tupla(string, string))
sendas(m) $\equiv$ compararTodos(estaciones(m), estaciones(m))

compararTodos : lista(estacion) $l1$ $\times$ lista(estacion) $l2$ $\times$ mapa m $\longrightarrow$ lista(tupla(string, string))
{estan?(l1, estaciones(m)) $\wedge$ estan?(l2, estaciones(m))}
compararTodos($l1, l2, m$) $\equiv$ **if** vacia?($l1$) **then**
<>
else
compararUno(prim($l1$), $l2$) & compararTodos(fin($l1$, $l2$))
fi
compararUno : estacion e $\times$ lista(estacion) l $\times$ mapa m $\longrightarrow$ lista(tupla(string, string))
{esta?(e, estaciones(m)) $\wedge$ estan?(l, estaciones(m))}
compararUno(e, l, m) $\equiv$ **if** vacia?(l) **then**
<>
else
if $e1$ >prim(l) $\wedge$ conectadas?($e1, prim$(l), m) **then**
<$e1, prim$(l)> $\bullet$ compararUno(e, fin(l))
else
compararUno(e, fin(l))
fi
fi
estan? : lista(α) $l1$ $\times$ lista(α) $l2$ $\longrightarrow$ bool
estan?($l1, l2$) $\equiv$ vacia?($l2$) $\vee_{\text{L}}$ ($\neg$vacia?($l1$) $\wedge_{\text{L}}$ (esta?(prim($l1$), $l2$) $\wedge$ estan?(fin($l1$), sacar(prim($l1$), $l2$))))

sacar : α e $\times$ lista(α) l $\longrightarrow$ bool
sacar(e, l) $\equiv$ **if** vacia?(l) **then** <> **else** **if** prim($l = e$) **then** sacar(e, fin(l)) **else** $e \bullet$ sacar(e, fin(l)) **fi** **fi**
Obs: α debe ser comparable con la funcion =.

esPermutacion? : lista(α) $l1$ $\times$ lista(α) $l2$ $\longrightarrow$ bool
esPermutacion?($l1, l2$) $\equiv$ estan?($l1, l2$) $\wedge$ estan?($l2, l1$)

1.2. Representacion

1.2.1. Representación de mapa

`map` se representa con `estr`

donde `estr` es `tupla`(*estaciones*: `lista(string)`,
sendas: `lista(tupla(e1: string, e2: string))`,
restricciones: $\texttt{dicc}_T\texttt{(dicc}_T\texttt{(restriccion))}$)

1.2.2. Invariante de Representación

(I) Las estaciones son las mismas que las claves de las sendas

(II) No esta definida una clave del diccionario dentro de sus diccionarios hijos

(III) Los diccionarios hijos de una clave estan definidos en el diccionario original

(IV) Las claves de los hijos diccionarios de un item del diccionario tienen menor orden lexicografico que el padre

(V) Las combinaciones definidas en las restricciones son las mismas que la lista de sendas

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$ ($\forall a$: string)($a \in e$.estaciones <=> def?(a, e.restricciones)) $\wedge$
($\forall c$, s: string)(def?(c, e.restricciones) $\Rightarrow_{\text{L}}$
($\neg$def?(c, obtener(c, e.restricciones)) $\wedge$
(def?(s, obtener(c, e.restricciones)) $\Rightarrow_{\text{L}}$ (def?(s, e) $\wedge$ $s < c$)))) $\wedge$
($\forall c$, s: string)(def?(c, e.restricciones) $\Rightarrow_{\text{L}}$
(def?(s, obtener(c, e.restricciones)) <=><c, s>$\in$ e.sendas))

1.2.3. Función de Abstracción

$$\text{Abs} : \text{estr } e \longrightarrow \text{mapa} \qquad \{\text{Rep}(e)\}$$

$$\begin{aligned}\text{Abs}(e) =_{\text{obs}} \text{m: mapa} \mid\ & \text{m.estaciones} =_{\text{obs}} \text{e.estaciones} \wedge \\ & (\forall c, s: \text{string})((c \in \text{estaciones}(e) \wedge s \in \text{estaciones}(e) \wedge c < s) \Rightarrow \\ & ((\text{def?}(s, \text{obtener}(c, e)) =_{\text{obs}} \text{conectadas?}(c, s, m)) \wedge_{\text{L}} \\ & (\text{def?}(s, \text{obtener}(c, e)) \Rightarrow_{\text{L}} (\text{obtener}(s, \text{obtener}(c, e)) =_{\text{obs}} \text{restriccion}(c, s, m)))))\end{aligned}$$

1.3. Algoritmos

iVacio () → res: estr

```
res.estaciones ← Vacia()                                   O(1)
res.sendas ← CrearDicc()                                   O(1)
```

Complejidad : $O(1)$

iAgregar (**in** e: `string`, **in/out** m: `estr`)

```
Agregar(e, m.estaciones)                                   O(1)
Definir(e, m.sendas)                                       O(|e|)
```

Complejidad : $O(|e|)$

iConectar (**in** $e1$: `string`, **in** $e2$: `string`, **in** r: `restriccion`, **in/out** m: `estr`)

```
if e1 < e2 then                                                          O(1)
  Conectar(e2, e1, r, m)                                                 O(|e1| + |e2|)
else
  if ¬Definido?(e1, m.restricciones) then                                O(|e1|)
    Definir(e1, CrearDicc(), m.restricciones)                            O(|e1|)
  end if
  Definir(e2, r, Obtener(e1, m.restricciones))                           O(|e1| + |e2|)
  AgregarAdelante(m.sendas, <e1, e2>)
end if
```

Complejidad : $O(|e1| + |e2|)$

iEsta? (**in** e: `string`, **in** m: `map`) → res: bool

```
res← Definido?(e, m.restricciones)                         O(|e|)
```

Complejidad : $O(|e|)$

iConectadas? (**in** $e1$: `string`, **in** $e2$: `string`, **in** m: `map`) → res: bool

```
res← Definido?(e1, m.restricciones) ∧L
     Definido?(e2, Obtener(e1, m.restricciones))           O(|e1| + |e2|)
```

Complejidad : $O(|e1| + |e2|)$

iRestriccion (**in** $e1$: `string`, **in** $e2$: `string`, **in** m : `map`) → res: restriccion

```
if e1 < e2 then                                                    O(1)
  Restriccion(e2, e1, m)                                           O(|e1| + |e2|)
else
  res← Obtener(e2, Obtener(e1, m.restricciones))                   O(|e1| + |e2|)
end if
```

Complejidad : $O(|e1| + |e2|)$

iEstaciones (**in** m : `map`) → res: itLista(string)

```
res← CrearIt(e.estaciones)                                         O(1)
```

Complejidad : $O(1)$

iSendas (**in** m : `map`) → res: itLista(tupla(string, string))

```
res← CrearIt(e.sendas)                                             O(1)
```

Complejidad : $O(1)$