```
*****************************************************************************
************                        Interfaz                         ************
*****************************************************************************

Se explica con: Mapa
Géneros: Mapa

*****************************************************************************
************                      Operaciones                        ************
*****************************************************************************

Crear() → res: Mapa
Pre ≡ {true}
Post ≡ {res =obs vacío}
Complejidad: O(1)
Descripcion: Crea un mapa vacío

Agregar(in/out m : Mapa, in e : estacion)
Pre ≡ {e ∉ estaciones(m) ∧ m₀ =obs m}
Post ≡ {m =obs agregar(e, m₀)}
Complejidad: O(#estaciones(m) + long(e))
Descripcion: Agrega una estación al mapa

Conectar(in/out m : Mapa, in e1 : estacion, in e2 : estacion, in r : restriccion)
Pre ≡ {e1, e2 ∈ estaciones(m) ∧ₗ ¬conectadas?(m, e1, e2) ∧ m₀ =obs m}
Post ≡ {m =obs conectar(e1, e2, r, m)}
Complejidad: O(long(e1) + long(e2) + S)
Descripcion: Conecta 2 estaciones del mapa entre con una restriccion para ese vinculo.

Estaciones(in m : Mapa) → res : itConj(string)
Pre ≡ {true}
Post ≡ {alias(esPermutación?(SecuSuby(res), estaciones(m))) ∧ vacia?(Anteriores(res))}
Complejidad: O(1)
Descripcion: Devuelve un iterador al conjunto de las estaciones del mapa.
Aliasing: No se pueden modificar los contenidos del iterador.

Conectadas?(in m : Mapa, in e1 : estacion, in e2 : estacion) → res : bool
Pre ≡ {e1, e2 ∈ estaciones(m)}
Post ≡ {res =obs conectadas?(m, e1, e2)}
Complejidad: O(long(e1) + long(e2))
Descripcion: Indica si 2 estaciones estan conectadas.

idSenda(in m : Mapa, in e1 : estacion, in e2 : estacion) → res : nat
Pre ≡ {e1, e2 ∈ estaciones(m) ∧ₗ conectadas?(m, e1, e2)}
Post ≡ {res es el id de la senda en nuestra representación}
Complejidad: O(long(e1) + long(e2))
Descripcion: Devuelve el id de la senda entre e1 y e2.

Sendas(in m : Mapa) → res : itVectorPointer(restriccion)
Pre ≡ {true}
Post ≡ {res itera sobre las restricciones de todas las sendas del mapa}
Complejidad: O(1)
Descripcion: Devuelve las restricciones de las sendas del mapa

Copiar(in m : Mapa) → res : Mapa
Pre ≡ {true}
Post ≡ {res =obs m}
Complejidad: O(Copiar(m.sendas) + Copiar(m.estaciones) + Copiar(m.conexiones))
Descripcion: Devuelve una copia del mapa.
Aliasing: El nuevo mapa comparte las restricciones ya agregadas al mapa original.

*****************************************************************************
************                     Representación                      ************
*****************************************************************************

Mapa se representa con estr,
    donde estr es: tupla(
                        sendas     : VectorPointer(Restriccion),
                        conexiones : DiccString(DiccString(Int)),
```

```
                        estaciones : conj(string)
                        )

Rep: ^(estr) → boolean
Rep(e) ≡ true ⟺
    claves(e.conexiones) ⊂ e.estaciones ∧ι
    ((∀e1 : string)
        e1 ∈ claves(e.conexiones) ∧ι
        claves(obtener(e1, e.conexiones)) ⊂ e.estaciones ∧ι
        ((∀e2 : string)
            (e1 ∈ claves(e.conexiones) ∧ι e2 ∈ claves(obtener(e1, e.conexiones)) ⟺
                e2 ∈ claves(e.conexiones) ∧ι e1 ∈ claves(obtener(e1, e.conexiones))) ∧ι
            (e1 ∈ claves(e.conexiones) ∧ι e2 ∈ claves(obtener(e1, e.conexiones)) ⟹ι
                obtener(e2, obtener(e1, e.conexiones)) < longitud(e.sendas))))

Abs: ^(estr) → mapa                                                    {Rep(e)}
(∀e : ^(estr)) Abs(e) =obs m /
    (e.estaciones =obs estaciones(m)) ∧
    ((∀ e1, e2: string)
        definido?(e.conexiones, e1) ∧ι
        (definido?(obtener(e.conexiones, e1), e2) =obs conectadas?(e1, e2, m)) ∧
        (conectadas?(e1, e2, m) ⟹ι *(e.sendas)[i] =obs restriccion(e1, e2, m)))

*****************************************************************************
************                      Algoritmos                       ************
*****************************************************************************

iCrear() → res : estr
    res ← ⟨sendas: Vacia(), conexiones: Vacio(), estaciones: Vacio()⟩
end function

iAgregar(in/out m : estr, in e : estacion)
    Agregar(m.estaciones, e)
end function

iConectar(in/out m : estr, in e1 : estacion, in e2 : estacion, in r : restriccion)
    var i : nat ← Longitud(m.sendas)
    AgregarAtras(m.sendas, &r)

    if ¬Definido?(m.conexiones, e1) then
        Definir(m.conexiones, e1, Vacio())
    end if

    if ¬Definido?(m.conexiones, e2) then
        Definir(m.conexiones, e2, Vacio())
    end if

    Definir(Obtener(m.conexiones, e1), e2, i)
    Definir(Obtener(m.conexiones, e2), e1, i)
end function

iEstaciones(in m : estr) → res : itConj(string)
    res ← CrearIt(m.estaciones)
end function

iConectadas?(in m : estr, in e1 : estacion, in e2 : estacion) → res : bool
    res ← Definido?(m.conexiones, e1) ∧ι Definido?(obtener(m.conexiones, e1), e2)
end function

iidSenda(in m : estr, in e1 : estacion, in e2 : estacion) → res : nat
    res ← Obtener(Obtener(m.conexiones, e1), e2)
end function

iSendas(in m : estr) → res : itVectorPointer(Restriccion)
    res ← CrearIt(m.sendas)
end function

iCopiar(in m : estr) → res : estr
    res ← ⟨    sendas     : Copiar(m.sendas),
            conexiones : Copiar(m.conexiones),
```

```
            estaciones : Copiar(m.estaciones)⟩
end function
```