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

Se explica con: conj(String)
Géneros: conjRapidoString

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

Vacio() → res : conjRapidoString
Pre ≡ {true}
Post ≡ {res =obs ϕ}
Complejidad: O(1)
Descripcion: Crea un conjunto vacio.

Agregar(in/out c : conjRapidoString, in s : string)
Pre ≡ {true}
Post ≡ {c =obs Ag(s, c)}
Complejidad: O(long(s))
Descripcion: Agrega un string al conjunto.

Pertenece?(in c : conjRapidoString, in s : string) → res : bool
Pre ≡ {true}
Post ≡ {res =obs s ∈ c}
Complejidad: O(long(s))
Descripcion: Se fija si s pertenece al conjunto o no.

CrearIt(in c : conjRapidoString) → res : itConj(string)
Pre ≡ {true}
Post ≡ {alias(esPermutación?(SecuSuby(res), c)) ∧ vacia?(Anteriores(res))}
Complejidad: O(1)
Descripcion: Devuelve un iterador del conjunto.

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

conjRapidoString se representa con conjRap, donde conjRap es DiccString(Bool)

Rep: ^(conjRap) → boolean
Rep(e) ≡ true ⟺ (∀s : string) Definido?(e, s) ⟹ι Obtener(e, s) =obs true

Abs: ^(conjRap) e → Conj(String)                                    {Rep(e)}
(∀e : ^(conjRap)) Abs(e) =obs c /
    (∀s : string) (Definido?(e, s) ∧ι Obtener(e, s)) =obs s ∈ c

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

iVacio() → res : conjRap
    res ← Crear()
end function

iAgregar(in/out c : conjRap, in s : string)
    if ¬Definido?(c, s) then
        Definir(c, s, true)
    end if
end function

iPertenece?(in c : conjRap, in s : string) → res : bool
    res ← Definido?(c, s)
end function

iCrearIt(in c : conjRap) → res : itConjString
    res ← Claves(c)
end function
```