

Algoritmos y Estructura de Datos II

Primer cuatrimestre 2014

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Practico 2

Grupo 10

Integrante	LU	Correo electrónico
Lucía, Parral	162/13	luciaparral@gmail.com
Nicolás, Roulet		
Pablo Nicolás, Gomez		
Guido Joaquin, Tamborindeguy		

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Módulo Wolfie	3
1.1. Interfaz	3
1.1.1. Parámetros formales	3
1.1.2. Operaciones básicas de wolfie	3
1.2. Representación	4
1.2.1. Representación de wolfie	4
2. Módulo DicionarioTrie(alpha)	6
2.1. Interfaz	6
2.1.1. Parámetros formales	6
2.1.2. Operaciones básicas de Dicionario String(α)	6
2.1.3. Operaciones básicas del iterador de claves de Dicionario String(α)	6
2.2. Representación	7
2.2.1. Representación del Dicionario String(α)	7
2.2.2. Operaciones auxiliares del invariante de Representación	7
2.2.3. Representación del iterador de Claves del Dicionario String(α)	8
2.3. Algoritmos	8
2.3.1. Algoritmos de Dicionario String	8
2.3.2. Algoritmos del iterador de claves del Dicionario String	9

1. Módulo Wolfie

1.1. Interfaz

1.1.1. Parámetros formales

géneros wolfie

se explica con: WOLFIE.

1.1.2. Operaciones básicas de wolfie

CLIENTES(**in** w : wolfie) $\rightarrow res$: itUni(cliente)
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{crearIt}(\text{clientes}(w))\}$
Complejidad: $\Theta(1)$
Descripcion: Devuelve un iterador a los clientes de un wolfie.

TÍTULOS(**in** w : wolfie) $\rightarrow res$: itUni(título)
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{crearItUni}(\text{títulos}(w))\}$
Complejidad: $\Theta(1)$
Descripcion: Devuelve un iterador a los títulos de un wolfie.

PROMESASDE(**in** c : cliente, **in** w : wolfie) $\rightarrow res$: itPromesa(promesa)
Pre $\equiv \{c \in \text{clientes}(w)\}$
Post $\equiv \{res =_{\text{obs}} \text{crearItUni}(\text{promesasDe}(c, w))\}$
Complejidad: $\Theta(T \cdot C \cdot |max_nt|)$
Descripcion: Devuelve un iterador a las promesas de un wolfie

ACCIONESPORCLIENTE(**in** c : cliente, **in** nt : nombreTítulo, **in** w : wolfie) $\rightarrow res$: nat
Pre $\equiv \{c \in \text{clientes}(w) \wedge (\exists t:\text{título}) (t \in \text{títulos}(w) \wedge \text{nombre}(t) = nt)\}$
Post $\equiv \{res =_{\text{obs}} \text{accionesPorCliente}(c, nt, w)\}$
Complejidad: $\Theta(\log(C) + |nt|)$
Descripcion: Devuelve la cantidad de acciones que un cliente posee de un determinado título.

INAUGURARWOLFIE(**in** cs : conj(cliente)) $\rightarrow res$: wolfie
Pre $\equiv \{\neg \emptyset?(cs)\}$
Post $\equiv \{res =_{\text{obs}} \text{inaugurarWolfie}(cs)\}$
Complejidad: $\Theta(\#(cs)^2)$
Descripcion: Crea un nuevo wolfie a partir de un conjunto de clientes.

AGREGARTÍTULO(**in** t : título, **in/out** w : wolfie) $\rightarrow res$: wolfie
Pre $\equiv \{w_0 =_{\text{obs}} w \wedge (\forall t2:\text{título}) (t2 \in \text{títulos}(w) \Rightarrow \text{nombre}(t) \neq \text{nombre}(t2))\}$
Post $\equiv \{w =_{\text{obs}} \text{agregarTítulo}(t, w_0)\}$
Complejidad: $\Theta(|\text{nombre}(t)| + C)$ **ACTUALIZARCOTIZACIÓN**(**in** nt : nombreTítulo, **in** cot : nat, **in/out** w : wolfie) $\rightarrow res$: wolfie
Pre $\equiv \{w_0 =_{\text{obs}} w \wedge (\exists t:\text{título}) (t \in \text{títulos}(w) \wedge \text{nombre}(t) = nt)\}$
Post $\equiv \{w =_{\text{obs}} \text{actualizarCotización}(nt, cot, w_0)\}$
Complejidad: $\Theta(C \cdot |nt| + C \cdot \log(C))$
Descripcion: Cambia la cotización de un determinado título. Esta operación genera que se desencadene el cumplimiento de promesas (según corresponda): primero de venta y luego, de compra, según el orden descendente de cantidad de acciones por título de cada cliente.

AGREGARPROMESA(**in** c : cliente, **in** p : promesa, **in/out** w : wolfie) $\rightarrow res$: wolfie
Pre $\equiv \{w_0 =_{\text{obs}} w \wedge (\exists t:\text{título}) (t \in \text{títulos}(w) \wedge \text{nombre}(t) = \text{título}(p)) \wedge c \in \text{clientes}(w) \wedge_L (\forall p2:\text{promesa}) (p2 \in \text{promesasDe}(c, w) \Rightarrow (\text{título}(p) \neq \text{título}(p2) \vee \text{tipo}(p) \neq \text{tipo}(p2))) \wedge (\text{tipo}(p) = \text{vender} \Rightarrow \text{accionesPorCliente}(c, \text{título}(p),$

$w) \geq \text{cantidad}(p)))\}$

Post $\equiv \{w =_{\text{obs}} \text{agregarPromesa}(c, p, w_0)\}$

Complejidad: $\Theta(|\text{título}(p)| + \log(C))$

Descripción: Agrega una nueva promesa.

ENALZA(in nt : nombreTítulo, in w : wolfie) $\rightarrow res$: bool

Pre $\equiv \{(\exists t: \text{título}) (t \in \text{títulos}(w) \wedge \text{nombre}(t) = nt)\}$

Post $\equiv \{res =_{\text{obs}} \text{enAlza}(nt, w)\}$

Complejidad: $\Theta(|nt|)$

Descripción: Dado un título, informa si está o no en alza.

1.2. Representación

1.2.1. Representación de wolfie

wolfie se representa con *estr*

donde *estr* es $\text{tupla}(\text{títulos: diccTrie}(\text{nombre}, \langle \text{arrayClientes: array_dimensionable}(\text{tuplaPorCliente}),$
 $\text{cot: nat},$
 $\text{enAlza: bool},$
 $\text{maxAcc: nat},$
 $\text{accDisponibles: nat}\rangle),$

$\text{clientes: conjEstNat}(\text{cliente})$

$\text{últimoLlamado: } \langle \text{cliente: cliente}, \text{promesas: conj}(\text{promesa}), \text{fueÚltimo: bool}\rangle)$

donde *tuplaPorCliente* es $\text{tupla}(\text{cliente: cliente}, \text{cantAcc: nat}, \text{promCompra: *promesa}, \text{promVenta: *promesa})$
 Con un orden definido por $a < b \Leftrightarrow a.\text{cliente} < b.\text{cliente}$

donde *tuplaPorCantAcc* es $\text{tupla}(\text{cliente: cliente}, \text{cantAcc: nat}, \text{promCompra: *promesa}, \text{promVenta: *promesa})$
 Con un orden definido por $a < b \Leftrightarrow a.\text{cantAcc} < b.\text{cantAcc}$

- (I) Los clientes de *clientes* son los mismos que hay dentro de *títulos*.
- (II) Las promesas de compra son de su título y cliente y no cumplen los requisitos para ejecutarse.
- (III) Las promesas de y venta son de su título y cliente y no cumplen los requisitos para ejecutarse.
- (IV) Las acciones disponibles de cada título son el máximo de acciones de ese título menos la suma de las acciones de ese título que tengan los clientes, y son mayores o iguales a 0.
- (V) El *cliente* de *últimoLlamado* pertenece a *clientes*.
- (VI) En *últimoLlamado*, si *fueÚltimo* es true, las promesas de *promesas* son todas las promesas que tiene *cliente*.
- (VII) Los clientes están ordenados en *arrayClientes* de *e.títulos*.

Rep : *estr* \rightarrow bool

$\text{Rep}(e) \equiv \text{true} \iff$
 (I) $(\forall c: \text{cliente}) \left(\text{pertenece?}(c, e.\text{clientes}) \iff (\exists t: \text{título}) \left(\text{def?}(t, e.\text{titulos}) \wedge_{\text{L}} \text{estáCliente?}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}) \right) \right) \wedge_{\text{L}}$
 (II) $(\forall p: *promesa, t: \text{nombre}, c: \text{cliente}) \left((p \neq \text{NULL} \wedge \text{def?}(t, e.\text{titulos}) \wedge_{\text{L}} \text{estáCliente?}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}) \wedge_{\text{L}} \text{buscarCliente}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).\text{promCompra}=p) \Rightarrow_{\text{L}} \text{título}(*p)=t \wedge \text{tipo}(*p)=\text{compra} \wedge (\text{límite}(*p) > \text{obtener}(t, e.\text{titulos}).\text{cot} \vee \text{cantidad}(*p) > \text{obtener}(t, e.\text{titulos}).\text{accDisponibles}) \right) \wedge$
 (III) $(\forall p: *promesa, t: \text{nombre}, c: \text{cliente}) \left((p \neq \text{NULL} \wedge \text{def?}(t, e.\text{titulos}) \wedge_{\text{L}} \text{estáCliente?}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}) \wedge_{\text{L}} \text{buscarCliente}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).\text{promVenta}=p) \Rightarrow_{\text{L}} (\text{título}(*p)=t \wedge \text{tipo}(*p)=\text{venta} \wedge \text{límite}(*p) < \text{obtener}(t, e.\text{titulos}).\text{cot}) \right) \wedge$
 (IV) $(\forall nt: \text{nombreT}) \left(\text{def?}(nt, e.\text{titulos}) \Rightarrow_{\text{L}} (\text{obtener}(nt, e.\text{titulos}).\text{accDisponibles} = \text{obtener}(nt, e.\text{titulos}).\text{maxAcc} - \text{sumaAccClientes}(\text{obtener}(nt, e.\text{titulos}).\text{arrayClientes}) \wedge \text{obtener}(nt, e.\text{titulos}).\text{accDisponibles} \geq 0) \right) \wedge$
 (V) $(\text{pertenece?}(e.\text{últimoLlamado}.cliente, e.\text{clientes})) \wedge_{\text{L}}$
 (VI) $(e.\text{últimoLlamado}.fueÚltimo \Rightarrow (\forall p: \text{promesa}) \left(\text{pertenece?}(p, e.\text{últimoLlamado}.promesas) \iff (\text{def?}(\text{título}(p), e.\text{titulos}) \wedge_{\text{L}} \right.$
 if $\text{tipo}(p)=\text{compra}$ **then**
 $\text{buscarCliente}(e.\text{últimoLlamado}.cliente, \text{obtener}(\text{título}(p), e.\text{titulos}).\text{arrayClientes}).\text{promCompra} = p$
 else
 $\text{buscarCliente}(e.\text{últimoLlamado}.cliente, \text{obtener}(\text{título}(p), e.\text{titulos}).\text{arrayClientes}).\text{promVenta} = p$
 fi $\left. \right) \wedge$
 (VII) $(\forall t: \text{título}) \text{def?}(t, e.\text{titulos}) \Rightarrow_{\text{L}} ((\forall i: \text{nat}) i < \text{longitud}(\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}) - 1 \Rightarrow (\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}[i] < (\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}[i+1])))$
 $\text{estáCliente?} : \text{cliente} \times \text{array_dimensionable}(\text{tuplaPorCliente}) \longrightarrow \text{bool}$
 $\text{estáCliente?}(c, a) \equiv \text{auxEstáCliente}(c, a, 0)$
 $\text{auxEstáCliente} : \text{cliente} \times \text{array_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{bool}$
 $\text{auxEstáCliente}(c, a, i) \equiv \text{if } i = \text{longitud}(a) \text{ then false else } a[i].cliente = c \vee \text{auxEstáCliente}(c, a, i+1) \text{ fi}$
 $\text{buscarCliente} : \text{cliente} \times \text{array_dimensionable}(\text{tuplaPorCliente}) \longrightarrow \text{tuplaPorCliente} \quad \{\text{estáCliente}(c, a)\}$
 $\text{buscarCliente}(c, a) \equiv \text{auxBuscarCliente}(c, a, 0)$
 $\text{auxBuscarCliente} : \text{cliente} \times \text{array_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{tuplaPorCliente} \quad \{\text{estáCliente}(c, a)\}$
 $\text{auxBuscarCliente}(c, a, i) \equiv \text{if } a[i].cliente = c \text{ then } a[i] \text{ else } \text{auxBuscarCliente}(c, a, i+1) \text{ fi}$
 $\text{Abs} : \text{estr } e \longrightarrow \text{wolfie} \quad \{\text{Rep}(e)\}$
 $\text{Abs}(e) =_{\text{obs}} w: \text{wolfie} \mid \text{clientes}(w) = e.\text{clientes} \wedge \text{títulos}(w) = \text{????????} \wedge$
 $(\forall c: \text{cliente}) \text{promesasDe}(c, w) = \text{damePromesas}(\text{crearIt}(e.\text{titulos}), e, c) \wedge$
 $\text{accionesPorCliente}(c, t, w) = \text{buscarCliente}(\text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).\text{cantAcc}$
 $\text{damePromesas} : \text{itTrie} \times \text{estr} \times \text{cliente} \longrightarrow \text{conj}(\text{promesa})$
 $\text{damePromesas}(it, e, c) \equiv \text{if hayMas?}(it) \text{ then}$
 if $\text{buscarCliente}(\text{obtener}(\text{actual}(it))).\text{promCompra} \neq \text{NULL}$ **then**
 $\{\text{buscarCliente}(\text{obtener}(\text{actual}(it))).\text{promCompra} \neq \text{NULL}\} \cup \text{fi}$
 if $\text{buscarCliente}(\text{obtener}(\text{actual}(it))).\text{promVenta} \neq \text{NULL}$ **then**
 $\{\text{buscarCliente}(\text{obtener}(\text{actual}(it))).\text{promVenta} \neq \text{NULL}\} \cup \text{fi}$
 $\text{damePromesas}(\text{avanzar}(it), e, c)$
 else
 vacío
 fi

2. Módulo DiccionarioTrie(alpha)

2.1. Interfaz

2.1.1. Parámetros formales

géneros string, α

se explica con: DICCIONARIO(String, α), ITERADOR UNIDIRECCIONAL.

géneros: diccString(α), itDicc(diccString).

2.1.2. Operaciones básicas de Diccionario String(α)

CREARDICC() $\rightarrow res : \text{diccString}(\alpha)$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} \text{vacío}\}$

Complejidad: $\Theta(1)$

Descripcion: Crea un diccionario vacío.

DEFINIR(in/out $d : \text{diccString}(\alpha)$, in $c : \text{string}$, in $s : \alpha$)

Pre $\equiv \{d =_{\text{obs}} d_0 \wedge \neg \text{def?}(d, c)\}$

Post $\equiv \{d =_{\text{obs}} \text{definir}(d_0, c, s)\}$

Complejidad: $\Theta(\text{longitud}(c))$

Descripcion: Define la clave c con el significado s en el diccionario d .

DEFINIDO?(in $d : \text{diccString}(\alpha)$, in $c : \text{string}$) $\rightarrow res : \text{bool}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} \text{def?}(c, d)\}$

Complejidad: $\Theta(\text{longitud}(c))$

Descripcion: Devuelve true si y solo si c está definido como clave en el diccionario.

SIGNIFICADO(in $d : \text{diccString}(\alpha)$, in $c : \text{string}$) $\rightarrow res : \alpha$

Pre $\equiv \{\text{def?}(c, d)\}$

Post $\equiv \{res =_{\text{obs}} \text{obtener}(c, d)\}$

Complejidad: $\Theta(\text{longitud}(c))$

Descripcion: Devuelve el significado con clave c .

Aliasing: No se devuelve una copia del α en res , se devuelve una referencia a la original.

2.1.3. Operaciones básicas del iterador de claves de Diccionario String(α)

CREARIT(in $d : \text{diccString}(\alpha)$) $\rightarrow res : \text{itClaves}(\text{string})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} \text{crearIt}(d.\text{claves})\}$

Complejidad: $\Theta(1)$

Descripcion: Crea y devuelve un iterador de claves Diccionario String.

HAYMAS?(in $d : \text{itClaves}(\text{string})$) $\rightarrow res : \text{bool}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} \text{hayMas?}(it)\}$

Complejidad: $\Theta(\text{longitud}(c))$

Descripcion: Informa si hay más elementos por iterar.

ACTUAL(in d : itClaves(string)) $\rightarrow res$: string
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{actual}(it)\}$
Complejidad: $\Theta(\text{longitud}(c))$
Descripcion: Devuelve la clave de la posición actual.

AVANZAR(in/out it : itClaves(string)) $\rightarrow res$: itClaves(α)
Pre $\equiv \{\text{hayMas?}(it) \wedge it = it_0\}$
Post $\equiv \{res =_{\text{obs}} \text{avanzar}(it_0)\}$
Complejidad: $\Theta(\text{longitud}(c))$
Descripcion: Avanza a la próxima clave.

2.2. Representacion

2.2.1. Representación del Diccionario String(α)

diccString(α) se representa con estrDic

donde **estrDic** es tupla(*raiz*: puntero(nodo) *claves*: lista_enlazada(string))

Nodo se representa con estrNodo

donde **estrNodo** es tupla(*valor*: puntero(α) *hijos*: arreglo_estatico[256](puntero(nodo)))

- (I) Existe un único camino entre cada nodo y el nodo raiz (es decir, no hay ciclos).
- (II) Todos los nodos hojas, es decir, todos los que tienen su arreglo hijos con todas sus posiciones en NULL, tienen que tener un valor distinto de NULL.
- (III) Raiz es distinto de NULL
- (IV) En claves está el camino que se recorre desde la raiz hasta cada nodo hoja.

Rep : estrDic \rightarrow bool

Rep(e) $\equiv \text{true} \iff$
 $\text{raiz} \neq \text{NULL} \wedge_{\text{L}} \text{noHayCiclos}(e) \wedge \text{todasLasHojasTienenValor}(e) \wedge$
 $\text{hayHojas}(e) \Rightarrow |e.\text{claves}| > 0 \wedge$
 $(\forall c \in \text{caminosANodos}(e))(\exists i \in \{0..|e.\text{claves}|\}) e.\text{claves}[i] = c$

Abs : estrDic $e \rightarrow \text{dicc}(\text{string}, \alpha)$

$\{\text{Rep}(e)\}$

Abs(e) $=_{\text{obs}} d$: $\text{dicc}(\text{string}, \alpha) \mid (\forall c:\text{string})(\text{definido?}(c, d)) = (\exists n:\text{nodo})(n \in \text{todasLasHojas}(e)) n.\text{valor} \neq \text{NULL}$
 $\wedge (\exists i \in \{0..|e.\text{claves}|\}) e.\text{claves}[i] = c \wedge_{\text{L}} \text{significado}(c, d) = \text{leer}(e.\text{clave}).\text{valor}$

2.2.2. Operaciones auxiliares del invariante de Representación

noHayCiclos : puntero(nodo) \rightarrow bool

noHayCiclos(n, p) $\equiv (\exists n:\text{nat})(\forall c:\text{string})(|s| = n \Rightarrow \text{leer}(p, s) = \text{NULL})$

leer : puntero(nodo) \times string \rightarrow bool

```

leer( $p, s$ )  $\equiv$  if vacia?( $s$ ) then
     $p \rightarrow \text{valor}$ 
else
    if  $p \rightarrow \text{hijos}[\text{prim}(s)] = \text{NULL}$  then NULL else leer( $p \rightarrow \text{hijos}[\text{prim}(s)], \text{fin}(s)$ ) fi
fi

todosNull : arreglo(puntero(nodo))  $\rightarrow$  bool
todosNull( $a$ )  $\equiv$  auxTodosNull( $a, 0$ )
auxTodosNull : arreglo(puntero(nodo))  $\times$  nat  $\rightarrow$  bool
auxTodosNull( $a, i$ )  $\equiv$  if  $i < |a|$  then  $a[i] == \text{NULL} \wedge \text{auxTodosNull}(a, i + 1)$  else  $a[i].\text{valor} == \text{NULL}$  fi

esHoja : puntero(nodo)  $\rightarrow$  bool
esHoja( $p$ )  $\equiv$  if  $p == \text{NULL}$  then false else todosNull( $p.\text{hijos}$ ) fi

todasLasHojas : puntero(nodo)  $\times$  nat  $\rightarrow$  conj(nodo)
todasLasHojas( $p, n$ )  $\equiv$  if  $p == \text{NULL}$  then
    false
else
    if esHoja( $p$ ) then Ag( $*p$ , vacio) else auxTodasLasHojas( $(*p).\text{hijos}, 256$ ) fi
fi

auxTodasLasHojas : arreglo(puntero(nodo))  $\times$  nat  $\rightarrow$  conj(nodo)
auxTodasLasHojas( $a, n$ )  $\equiv$  hojasDeHijos( $a, n, 0$ )
hojasDeHijos : arreglo(puntero(nodo))  $\times$  nat  $\times$  nat  $\rightarrow$  conj(nodo)
hojasDeHijos( $a, n, i$ )  $\equiv$  if  $i = n$  then  $\emptyset$  else todasLasHojas( $a[i]$ )  $\cup$  hojasDeHijos( $a, n, (i + 1)$ ) fi

todasLasHojasTienenValor : puntero(nodo)  $\rightarrow$  bool
todasLasHojasTienenValor( $p$ )  $\equiv$  auxTodasLasHojasTienenValor(todasLasHojas( $p, 256$ ))
auxTodasLasHojasTienenValor : arreglo(puntero(nodo))  $\rightarrow$  bool
auxTodasLasHojasTienenValor( $a$ )  $\equiv$  if  $|a| = 0$  then
    true
else
    dameUno( $a$ ).valor  $\neq \text{NULL} \wedge \text{auxTodasLasHojasTienenValor}(\text{sinUno}(a))$ 
fi

```

2.2.3. Representación del iterador de Claves del Diccionario String(α)

itClaves(*string*) se representa con puntero(nodo)

Su Rep y Abs son los de itSecu(α) definido en el apunte de iteradores..

2.3. Algoritmos

2.3.1. Algoritmos de Diccionario String

```

iCrearDicc()  $\rightarrow$  res = diccString( $\alpha$ )
1   $n \leftarrow \text{nodo}$ 
2   $n \leftarrow \text{crearNodo}()$ 
3   $\text{raiz} \leftarrow *n$ 

iCrearNodo()  $\rightarrow$  res = nodo
1   $d : \text{arreglo} \setminus \_ \text{estatico}[256]$ 
2   $i \leftarrow 0$ 
3  while ( $i < 256$ )
4       $d[i] \leftarrow \text{NULL}$ 

```



```

5  endWhile
6  hijos ← d
7  valor ← NULL

iDefinir(inout diccString( $\alpha$ ): d, in string: c, in alfa: s)
1  i ← 0
2  p ← d.raiz
3  while (i < (longitud(s)))
4      if (p.hijos[ord(s[i])] == NULL)
5          n: nodo ← crearNodo()
6          p.hijos[ord(s[i])] ← *n
7      endIf
8  p ← p.hijos[ord(s[i])]
9  i++
10 endWhile
11 p.valor ← a
12 agregarAdelante(hijos, c)

iSignificado(in diccString( $\alpha$ ): d, in string: c) → res = alfa
1  i ← 0
2  p ← d.raiz
3  while (i < (longitud(s)))
4      p ← p.hijos[ord(s[i])]
5  i++
6  endWhile
7  return p.valor

iDefinido?(in diccString: d, in string: c) → res = bool
1  i ← 0
2  p ← d.raiz
3  while (i < (longitud(s)))
4      if (p.hijos[ord(s[i])] != NULL)
5          p ← p.hijos[ord(s[i])]
6          i++
7      else
8          return false
9      endIf
10 endWhile
11 return p.valor != NULL

iClaves(in diccString: d) → res = lista_enlazada(string)
1 return itClaves(d)

```

2.3.2. Algoritmos del iterador de claves del Diccionario String

Utiliza los mismos algoritmos que $\text{itSecu}(\alpha)$ definido en el apunte de iteradores.