

# 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

<b>1. Renombres de Módulos</b>	<b>3</b>
<b>2. Módulo Wolfie</b>	<b>3</b>
2.1. Interfaz	3
2.1.1. Parámetros formales	3
2.1.2. Operaciones básicas de wolfie	3
2.2. Representación	4
2.2.1. Representación de wolfie	4
<b>3. Módulo DicionarioTrie(alpha)</b>	<b>6</b>
3.1. Interfaz	6
3.1.1. Parámetros formales	6
3.1.2. Operaciones básicas de Dicionario String( $\alpha$ )	7
3.1.3. Operaciones básicas del iterador de claves de Dicionario String( $\alpha$ )	7
3.2. Representación	8
3.2.1. Representación del Dicionario String( $\alpha$ )	8
3.2.2. Operaciones auxiliares del invariante de Representación	8
3.2.3. Representación del iterador de Claves del Dicionario String( $\alpha$ )	9
3.3. Algoritmos	9
3.3.1. Algoritmos de Dicionario String	9
3.3.2. Algoritmos del iterador de claves del Dicionario String	10
<b>4. Módulo Conjunto Estático de Nats</b>	<b>10</b>
4.1. Interfaz	10
4.1.1. Operaciones básicas de conjEstNat	11
4.1.2. Operaciones básicas de itConjEstNat	11
4.2. Representación	12
4.2.1. Representación de conjEstNat	12
4.2.2. Representación de itConjEstNat	12
<b>5. Módulo Promesa</b>	<b>13</b>
5.1. Interfaz	13
5.1.1. Parámetros formales	13
5.1.2. Operaciones básicas de promesa	13
5.2. Representación	14
5.2.1. Representación de promesa	14
5.3. Algoritmos	14
5.3.1. Algoritmos de promesa	14

## 1. Renombres de Módulos

Módulo Dinero es Nat  
 Módulo Cliente es Nat  
 Módulo TipoPromesa es enum{compra, venta}  
 Módulo Nombre es String

## 2. Módulo Wolfie

### 2.1. Interfaz

#### 2.1.1. Parámetros formales

**géneros**    wolfie

**se explica con:** WOLFIE.

#### 2.1.2. Operaciones básicas de wolfie

**CLIENTES**(in  $w$ : wolfie)  $\rightarrow res$ : itConjEstNat(cliente)  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{crearItUni}(\text{clientes}(w))\}$   
**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)  $\rightarrow res$ : itConj(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)  
**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:**  $\Theta(C \cdot |nt| + C \cdot \log(C))$

**Descripción:** 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)

**Pre**  $\equiv \{w_0 =_{\text{obs}} w \wedge (\exists t: \text{título}) (t \in \text{títulos}(w) \wedge \text{nombre}(t) = \text{título}(p)) \wedge c \in \text{clientes}(w) \wedge (\forall p2: \text{promesa}) (p2 \in \text{promesasDe}(c, w) \Rightarrow (\text{título}(p) \neq \text{título}(p2) \vee \text{tipo}(p) \neq \text{tipo}(p2))) \wedge (\text{tipo}(p) = \text{vender} \Rightarrow \text{accionesPorCliente}(c, \text{título}(p), w) \geq \text{cantidad}(p)))\}$

**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.

## 2.2. Representación

### 2.2.1. Representación de wolfie

wolfie se representa con *estr*

donde *estr* es *tupla*(*títulos*: diccTrie(nombre, <arrayClientes: array\_dimensionable(tuplaPorCliente),  
*cot*: nat,  
*enAlza*: bool,  
*maxAcc*: nat,  
*accDisponibles*: nat>),  
*clientes*: conjEstNat(cliente)  
*últimoLlamado*: <cliente: cliente, promesas: conj(promesa), fueÚltimo: bool>))

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

donde *tuplaPorCantAcc* es *tupla*(*cliente*: cliente, *cantAcc*: nat, *promCompra*: \*promesa, *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)  $(\forall nt: \text{nombreT}) \left( \text{def?}(nt, e.\text{titulos}) \Rightarrow_{\text{L}} (\text{obtener}(nt, e.\text{titulos}).\text{accDisponibles} = \text{obtener}(nt, e.\text{titulos}).\text{maxAcc} - \text{sumaAccClientes}(\text{obtener}(nt, e.\text{titulos}).\text{arrayClientes}, 0) \wedge \text{obtener}(nt, e.\text{titulos}).\text{accDisponibles} \geq 0) \right) \wedge$   
 (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**  $\left. \right) \wedge_{\text{L}}$   
 (VII)  $(\forall t: \text{título}) \text{def?}(t, e.\text{titulos}) \Rightarrow_{\text{L}} ((\forall i: \text{nat}) i < \text{longitud}(\text{buscar}(t, e.\text{titulos}).\text{arrayClientes})-1 \Rightarrow (\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}[i] < (\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}[i+1])))$   
 $\text{estáCliente?} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \longrightarrow \text{bool}$   
 $\text{estáCliente?}(c, a) \equiv \text{auxEstáCliente}(c, a, 0)$   
 $\text{auxEstáCliente} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{bool}$   
 $\text{auxEstáCliente}(c, a, i) \equiv \text{if } i=\text{longitud}(a) \text{ then false else } a[i].cliente = c \vee \text{auxEstáCliente}(c, a, i+1) \text{ fi}$   
 $\text{buscarCliente} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \longrightarrow \text{tuplaPorCliente} \quad \{\text{estáCliente}(c, a)\}$   
 $\text{buscarCliente}(c, a) \equiv \text{auxBuscarCliente}(c, a, 0)$   
 $\text{auxBuscarCliente} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{tuplaPorCliente} \quad \{\text{estáCliente}(c, a)\}$   
 $\text{auxBuscarCliente}(c, a, i) \equiv \text{if } a[i].cliente = c \text{ then } a[i] \text{ else } \text{auxBuscarCliente}(c, a, i+1) \text{ fi}$   
 $\text{sumaAccClientes} : \text{array\_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{nat}$   
 $\text{auxBuscarCliente}(a, i) \equiv \text{if } i=\text{longitud}(a) \text{ then } 0 \text{ else } a[i].cantAcc + \text{sumaAccClientes}(a, i+1) \text{ fi}$   
 $\text{Abs} : \text{estr } e \longrightarrow \text{wolfie} \quad \{\text{Rep}(e)\}$   
 $\text{Abs}(e) =_{\text{obs}} w: \text{wolfie} \mid \text{clientes}(w)=e.\text{clientes} \wedge \text{títulos}(w)=???????? \wedge$   
      $(\forall c: \text{cliente}) \text{promesasDe}(c, w)=\text{damePromesas}(\text{crearIt}(e.\text{titulos}), e, c) \wedge$   
      $\text{accionesPorCliente}(c, t, w)=\text{buscarCliente}(\text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).cantAcc$   
 $\text{damePromesas} : \text{itDicc}(\text{diccString}) \times \text{estr} \times \text{cliente} \longrightarrow \text{conj}(\text{promesa})$

```

damePromesas(it, e, c)  $\equiv$  if hayMas?(it) then
    if buscarCliente(obtener(actual(it))).promCompra  $\neq$  NULL then
        {buscarCliente(obtener(actual(it))).promCompra  $\neq$  NULL}  $\cup$  fi
    if buscarCliente(obtener(actual(it))).promVenta  $\neq$  NULL then
        {buscarCliente(obtener(actual(it))).promVenta  $\neq$  NULL}  $\cup$  fi
    damePromesas(avanzar(it), e, c)
else
    vacio
fi

iClientes(in e: estr)  $\rightarrow$  res = itConjEstNat
1  return ( CrearIt ( e. clientes ) )

iPromesasDe(in c: cliente, in/out e: estr)  $\rightarrow$  res = itConj(promesa)
1  if  $\neg$ (e. ultimoLlamado. cliente = c  $\wedge$  e. ultimoLlamado. fueUltimo) then
2      itClaves(diccString) it  $\leftarrow$  crearIt ( e. titulos )
3      conj(promesa) proms  $\leftarrow$  vacio ()
4      tuplaPorClientes tup
5      while ( HayMas?( it ) )
6          tup  $\leftarrow$  BuscarCliente( Obtener( Nombre( Actual( it ) ), e. titulos ). arrayClientes )
7          if tup.promVenta  $\neq$  NULL then AgregarRapido( proms, *(tup.promVenta))
8          if tup.promCompra  $\neq$  NULL then AgregarRapido( proms, *(tup.promCompra))
9          Avanzar( it )
10     endWhile
11     e. ultimoLlamado. promesas  $\leftarrow$  proms
12 fi
13 return ( crearIt ( e. ultimoLlamado. promesas ) )

iAccionesPorCliente(in c: cliente, in nt, nombreT, in e: estr)  $\rightarrow$  res = nat
1  return ( BuscarCliente( c , Obtener( nt , e. titulos ) ). cantAcc )

iInaugurarWolfie(in c: conj(cliente))  $\rightarrow$  res = estr
1  res. titulos  $\leftarrow$  CrearDicc ()
2  res. clientes  $\leftarrow$  CrearDicc ()
3  res. ultimoLlamado  $\leftarrow$  <0, Vacio (), false>

iAgregarPromesa(in c: cliente, in p:promesa, in/out e:estr)
1  promesa prom  $\leftarrow$  p
2  if tipo(prom)=compra then
3      BuscarCliente( c , Obtener( titulo (prom), e. titulos ). arrayClientes ).promCompra  $\leftarrow$  &prom
4  else
5      BuscarCliente( c , Obtener( titulo (prom), e. titulos ). arrayClientes ).promCompra  $\leftarrow$  &prom
6  fi

iEnAlza(in nt: nombreT, in e: estr)  $\rightarrow$  res = bool
1  return ( Obtener( nt , e. titulos ). enAlza )

iAgregarTitulo(in t: titulo, in/out e: estr)  $\rightarrow$  res = nat
1  Definir ( e. titulos , nombre( t ) , <CrearArrayClientes

```

### 3. Módulo DiccionarioTrie(alpha)

#### 3.1. Interfaz

##### 3.1.1. Parámetros formales

**géneros**    string,  $\alpha$

**se explica con:** DICCIONARIO( $\alpha$ ), ITERADOR UNIDIRECCIONAL.

**géneros:** diccString( $\alpha$ ), itDicc(diccString).

### 3.1.2. Operaciones básicas de Diccionario String( $\alpha$ )

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

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

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

**Complejidad:**  $O(1)$

**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:**  $O(\text{longitud}(c))$

**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:**  $O(\text{longitud}(c))$

**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:**  $O(\text{longitud}(c))$

**Descripcion:** Devuelve el significado con clave  $c$ .

**Aliasing:** No se devuelve una copia del  $\alpha$  en  $res$ , se devuelve una referencia a la original.

### 3.1.3. Operaciones básicas del iterador de claves de Diccionario String( $\alpha$ )

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:**  $O(1)$

**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:**  $O(\text{longitud}(c))$

**Descripcion:** Informa si hay más elementos por iterar.

ACTUAL(**in**  $d : \text{itClaves}(\text{string})$ )  $\rightarrow res : \text{string}$

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

**Post**  $\equiv \{res =_{\text{obs}} \text{actual}(it)\}$

**Complejidad:**  $O(\text{longitud}(c))$

**Descripcion:** Devuelve la clave de la posición actual.

AVANZAR(**in/out**  $it : \text{itClaves}(\text{string})$ )  $\rightarrow res : \text{itClaves}(\alpha)$

**Pre**  $\equiv \{\text{hayMas?}(it) \wedge it = it_0\}$

**Post**  $\equiv \{\text{res} =_{\text{obs}} \text{avanzar}(it_0)\}$

**Complejidad:**  $O(\text{longitud}(c))$

**Descripcion:** Avanza a la próxima clave.

## 3.2. Representacion

### 3.2.1. Representación del Diccionario String( $\alpha$ )

$\text{diccString}(\alpha)$  se **representa con**  $\text{estrDic}$

donde  $\text{estrDic}$  es  $\text{tupla}(\text{raiz: puntero(nodo)} \quad \text{claves: lista\_enlazada(string)})$

Nodo se **representa con**  $\text{estrNodo}$

donde  $\text{estrNodo}$  es  $\text{tupla}(\text{valor: puntero}(\alpha) \quad \text{hijos: arreglo\_estatico}[256](\text{puntero(nodo)}) )$

- (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.

$\text{Rep} : \text{estrDic} \longrightarrow \text{bool}$

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

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

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

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

### 3.2.2. Operaciones auxiliares del invariante de Representación

$\text{noHayCiclos} : \text{puntero(nodo)} \longrightarrow \text{bool}$

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

$\text{leer} : \text{puntero(nodo)} \times \text{string} \longrightarrow \text{bool}$

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

$\text{todosNull} : \text{arreglo(puntero(nodo))} \longrightarrow \text{bool}$

$\text{todosNull}(a) \equiv \text{auxTodosNull}(a, 0)$

$\text{auxTodosNull} : \text{arreglo(puntero(nodo))} \times \text{nat} \longrightarrow \text{bool}$

$\text{auxTodosNull}(a, i) \equiv \text{if } i < |a| \text{ then } a[i] == \text{NULL} \wedge \text{auxTodosNull}(a, i + 1) \text{ else } a[i].\text{valor} == \text{NULL} \text{ fi}$



```

esHoja : puntero(nodo)  → bool
esHoja(p) ≡ if p == NULL then false else todosNull(p.hijos) fi
todasLasHojas : puntero(nodo) × nat  → conj(nodo)
todasLasHojas(p, n) ≡ if p == NULL then
    false
    else
        if esHoja(p) then Ag(*p, vacio) else auxTodasLasHojas((*p).hijos, 256) fi
    fi
auxTodasLasHojas : arreglo(puntero(nodo)) × nat  → conj(nodo)
auxTodasLasHojas(a, n) ≡ hojasDeHijos(a, n, 0)
hojasDeHijos : arreglo(puntero(nodo)) × nat × nat  → conj(nodo)
hojasDeHijos(a, n, i) ≡ if i = n then ∅ else todasLasHojas(a[i] ∪ hojasDeHijos(a, n, (i + 1)) fi
todasLasHojasTienenValor : puntero(nodo)  → bool
todasLasHojasTienenValor(p) ≡ auxTodasLasHojasTienenValor(todasLasHojas(p, 256))
auxTodasLasHojasTienenValor : arreglo(puntero(nodo))  → bool
auxTodasLasHojasTienenValor(a) ≡ if |a| = 0 then
    true
    else
        dameUno(a).valor != NULL ∧ auxTodasLasHojasTienenValor(sinUno(a))
    fi

```

### 3.2.3. Representación del iterador de Claves del Diccionario String( $\alpha$ )

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

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

## 3.3. Algoritmos

### 3.3.1. Algoritmos de Diccionario String

```

ICREARDICC() → res = diccString( $\alpha$ )
1  n ← nodo
2  n ← crearNodo()
3  raiz ← *n

```

#### Complejidad

```

ICREARNODO() → res = nodo
1  d : arreglo\_estatico[256]
2  i ← 0
3  while (i < 256)
4      d[i] ← NULL
5  endWhile
6  hijos ← d
7  valor ← NULL

```

#### Complejidad

```

IDEFINIR(in/out 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)

```

### Complejidad

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

```

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

```

### Complejidad

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

```

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

```

### Complejidad

$\text{ICLAVES}(\text{in diccString}(\alpha): d) \rightarrow \text{res} = \text{lista\_enlazada}(\text{string})$

```

1 return itClaves(d)

```

### Complejidad

#### 3.3.2. Algoritmos del iterador de claves del Diccionario String

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

## 4. Módulo Conjunto Estático de Nats

### 4.1. Interfaz

**géneros**    conjEstNat, itConjEstNat

**Se explica con:**  $\text{CONJUNTO}(\text{NAT})$ ,  $\text{ITERADOR UNIDIRECCIONAL}(\text{NAT})$ . **Usa:**

#### 4.1.1. Operaciones básicas de conjEstNat

**NUEVOCONJESTNAT**(in  $c: \text{conj}(\text{nat})$ )  $\rightarrow res: \text{conjEstNat}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} c\}$   
**Complejidad:**  $\Theta(n * (\log(n)))$   
**Descripcion:** Crea un conjunto estático de nats

**PERTENECE?**(in  $n: \text{nat}$ , in  $c: \text{conjEstNat}$ )  $\rightarrow res: \text{bool}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} n \in c\}$   
**Complejidad:**  $\Theta(n)$   
**Descripcion:** Pregunta si el elemento pertenece al conjunto

**CARDINAL**(in  $n: \text{conjEstNat}$ )  $\rightarrow res: \text{nat}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} n \in c\}$   
**Complejidad:**  $\Theta(n)$   
**Descripcion:** Pregunta si el elemento pertenece al conjunto

#### 4.1.2. Operaciones básicas de itConjEstNat

**CREARIT**(in  $c: \text{conjEstNat}$ )  $\rightarrow res: \text{itConjEstNat}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{crearItUni}(c)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripcion:** Devuelve un iterador unidireccional a un conjunto estático de nats

**ACTUAL**(in  $i: \text{itConjEstNat}$ )  $\rightarrow res: \text{nat}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{actual}(i)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripcion:** Devuelve la posicion actual

**PRÓXIMO**(in  $i: \text{itConjEstNat}$ )  $\rightarrow res: \text{itConjEstNat}$   
**Pre**  $\equiv \{\text{hayMas?}(i)\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{avanzar}(i)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripcion:** Avanza el iterador

**HAYPRÓX?**(in  $i: \text{itConjEstNat}$ )  $\rightarrow res: \text{bool}$   
**Pre**  $\equiv \{i_0 = i\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{hayMas?}(i)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripcion:** Pregunta si hay mas elementos para iterar

## 4.2. Representación

### 4.2.1. Representación de conjEstNat

conjEstNat se representa con array: arreglo\_dimensionable(nat)

Rep: los elementos estan ordenados y no hay repeticiones

Rep : array  $\rightarrow$  bool

Rep( $a$ )  $\equiv$  true  $\iff (\forall i: \text{nat}) (i < \text{longitud}(a)-1 \Rightarrow (\text{definido?}(a, i) \wedge \text{definido?}(a, i+1) \wedge_L a[i] < a[i+1]))$

Abs : array  $a \rightarrow$  conjEstNat

{Rep( $a$ )}

Abs( $a$ ) =<sub>obs</sub>  $c$ : conjEstNat |  $(\forall n: \text{nat}) n \in c \Leftrightarrow \text{estáEnArray?}(n, a, 0)$

estáEnArray? : nat  $\times$  arreglo\_dimensionable(nat)  $\times$  nat  $\rightarrow$  bool

estáEnArray( $n, a, i$ )  $\equiv$  **if**  $i = \text{longitud}(a)-1$  **then** false **else**  $a[i] = n \vee \text{estáEnArray?}(n, a, i+1)$  **fi**

### 4.2.2. Representación de itConjEstNat

itConjEstNat se representa con iterador

donde iterador es tupla( $pos$ : nat,  $lista$ : puntero(arreglo\_dimensionable(nat)) )

Rep : iterador  $\rightarrow$  bool

Rep( $i$ )  $\equiv$  true  $\iff i.pos < \text{longitud}(*i.lista)$

Abs : iterador  $it \rightarrow$  itConjEstNat

{Rep( $it$ )}

Abs( $it$ ) =<sub>obs</sub>  $iConj$ : itConjEstNat |  $\text{actual}(iConj) = a[i] \wedge \text{hayPróx}(iConj) = (i.pos < \text{longitud}(*i.lista)-1) \wedge (\text{hayPróx}(iConj) \Rightarrow \text{próximo}(iConj) = \text{Abs}(<i.pos + 1, i.lista>))$

iNuevoConjEstNat(in  $c$ : conj(nat))  $\rightarrow$  res = array

```

1 itConj(nat)  $it \leftarrow \text{crearIt}(c)$ 
2 arreglo_dimensionable(nat)[cardinal(c)]  $a$ 
3 nat  $i \leftarrow 0$ 
4 while ( $\text{HaySiguiente?}(it)$ )
5      $a[i] \leftarrow \text{Siguiente}(it)$ 
6      $i++$ 
7      $\text{Avanzar}(it)$ 
8 endWhile
9  $\text{heapsort}(a)$ 
10 return ( $a$ )

```

iPertenece(in  $n$ : nat, in  $c$ : array)  $\rightarrow$  res = bool

```

1 bool  $b \leftarrow \text{false}$ 
2 nat  $i \leftarrow 0$ 
3 while ( $i < |c|$ )
4      $b \leftarrow (b \vee c[i] = n)$ 
5      $i++$ 
6 endWhile
7 return ( $b$ )

```

iCrearIt(in  $a$ : array)  $\rightarrow$  res = iterador

```

1 return ( $<|c|, \&c>$ )

```

iActual(in  $it$ : iterador)  $\rightarrow$  res = nat

```

1 return  $*(it.lista)[it.pos]$ 

```

iActual(in/out  $it$ : iterador)

```

1 return <it.pos+1, it.lista>

iHayPróximo?(in it: iterador) → res = bool
1 return (it.pos+1<longitud(it.lista))

```

Servicios usados: se utilizan solo tipos básicos, incluidos arreglos y punteros.

## 5. Módulo Promesa

### 5.1. Interfaz

#### 5.1.1. Parámetros formales

**géneros**    promesa

**se explica con:** PROMESA.

#### 5.1.2. Operaciones básicas de promesa

**TÍTULO**(in  $p$ : promesa) →  $res$  : nombre  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{título}(p)\}$   
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve el nombre del título de la promesa

**TIPO**(in  $p$ : promesa) →  $res$  : tipoPromesa  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{tipo}(p)\}$   
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve el tipo de promesa de la promesa

**LÍMITE**(in  $p$ : promesa) →  $res$  : dinero  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{límite}(p)\}$   
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve el límite de la promesa

**CANTIDAD**(in  $p$ : promesa) →  $res$  : cantidad  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{cantidad}(p)\}$   
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve la cantidad de acciones de la promesa

**CREARPROMESA**(in  $t$ : nombre, in  $tipo$ : tipoPromesa, in  $n$ : dinero, in  $m$ : nat) →  $res$  : estr  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{crearPromesa}(t, tipo, n, m)\}$   
**Complejidad:** (1)  
**Descripcion:** Devuelve una nueva promesa

## 5.2. Representación

### 5.2.1. Representación de promesa

**promesa se representa con estr**

donde **estr** es  $\text{tupla}(\text{título: nombre } \text{tipo: tipoPromesa } \text{límite: dinero } \text{cantidad: nat})$

$\text{Rep} : \text{estr} \rightarrow \text{bool}$

$\text{Rep}(e) \equiv \text{true} \iff \text{true}$

$\text{Abs} : \text{estr } e \rightarrow \text{promesa}$

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

$\text{Abs}(e) =_{\text{obs}} p : \text{promesa} \mid \text{título}(p) = e.\text{título} \wedge \text{tipo}(p) = e.\text{tipo} \wedge \text{límite}(p) = e.\text{límite} \wedge \text{cantidad}(p) = e.\text{cantidad}$

## 5.3. Algoritmos

### 5.3.1. Algoritmos de promesa

$\text{iTitulo}(\text{in } p : \text{estr}) \rightarrow \text{res} = \text{nombre}$

1  $\text{res} = e.\text{titulo}$

$\text{iTipo}(\text{in } p : \text{estr}) \rightarrow \text{res} = \text{tipoPromesa}$

1  $\text{res} = e.\text{tipo}$

$\text{iLimite}(\text{in } p : \text{estr}) \rightarrow \text{res} = \text{dinero}$

1  $\text{res} = e.\text{limite}$

$\text{iCantidad}(\text{in } p : \text{estr}) \rightarrow \text{res} = \text{nat}$

1  $\text{res} = e.\text{cantidad}$

$\text{iCrearPromesa}(\text{in } t : \text{nombreT}, \text{in } \text{tipo} : \text{TipoPromesa}, \text{in } n : \text{dinero}, \text{in } c : \text{nat}) \rightarrow \text{res} = \text{estr}$

1  $\text{res.titulo} = t$

2  $\text{res.tipo} = \text{tipo}$

3  $\text{res.limite} = n$

4  $\text{res.cantidad} = m$