



DEPARTAMENTO  
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

## Trabajo Practico 2

---

Algoritmos y Estructura de Datos II  
Primer cuatrimestre 2014

### Grupo 10

Integrante	LU	Correo electrónico
Gómez, Pablo Nicolás	156/13	mag0-1986@hotmail.com
Parral, Lucía Inés	162/13	luciaparral@gmail.com
Roulet, Nicolás	587/13	nicoroulet@gmail.com
Tamborindeguy, Guido Joaquín	584/13	guido@tamborindeguy.com.ar



**Facultad de Ciencias Exactas y Naturales**  
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.fcen.uba.ar>

# 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
2.1.3. Operaciones básicas de itTítulos	4
2.2. Representación	5
2.2.1. Representación de wolfie	5
2.2.2. Invariante de representación	5
2.2.3. Función de abstracción	6
2.2.4. Representación de itTítulos	7
2.2.5. Invariante de Representación de itTítulos	7
2.3. Algoritmos	7
2.3.1. Algoritmos de wolfie	7
2.3.2. Algoritmos de itTítulos	9
2.3.3. Funciones auxiliares	9
2.4. Servicios Usados	10
<b>3. Módulo Diccionario String(<math>\alpha</math>)</b>	<b>10</b>
3.1. Interfaz	10
3.1.1. Parámetros formales	10
3.1.2. Operaciones básicas de Diccionario String( $\alpha$ )	10
3.1.3. Operaciones básicas del iterador de claves de Diccionario String( $\alpha$ )	11
3.2. Representación	12
3.2.1. Representación del Diccionario String( $\alpha$ )	12
3.2.2. Invariante de Representación de diccString	12
3.2.3. Operaciones auxiliares del invariante de Representación	12
3.2.4. Función de abstracción de diccString	13
3.2.5. Representación del iterador de Claves del Diccionario String( $\alpha$ )	13
3.3. Algoritmos	13
3.3.1. Algoritmos de Diccionario String	13
3.3.2. Algoritmos del iterador de claves del Diccionario String	14
3.4. Servicios Usados	14
<b>4. Módulo Conjunto Estático de Nats</b>	<b>14</b>
4.1. Interfaz	14
4.1.1. Operaciones básicas de conjEstNat	15
4.1.2. Operaciones básicas de itConjEstNat	15
4.2. Representación	16
4.2.1. Representación de conjEstNat	16

4.2.2.	Función de abstracción de conjEstNat	16
4.2.3.	Representación de itConjEstNat	16
4.2.4.	Función de abstracción de itConjEstNat	16
4.3.	Algoritmos	16
4.3.1.	Algoritmos de conjEstNat	16
4.3.2.	Algoritmos de itConjEstNat	17
4.4.	Servicios Usados	17
4.5.	TAD CONJUNTO ESTÁTICO DE NATS	17
<b>5.</b>	<b>Módulo Promesa</b>	<b>18</b>
5.1.	Interfaz	18
5.1.1.	Parámetros formales	18
5.1.2.	Operaciones básicas de promesa	18
5.2.	Representación	19
5.2.1.	Representación de promesa	19
5.2.2.	Abs	19
5.3.	Algoritmos	19
5.3.1.	Algoritmos de promesa	19
<b>6.</b>	<b>Módulo Título</b>	<b>20</b>
6.1.	Interfaz	20
6.1.1.	Parámetros formales	20
6.1.2.	Operaciones básicas de título	20
<b>7.</b>	<b>Módulo Wolfie</b>	<b>21</b>
7.1.	Interfaz	21
7.1.1.	Parámetros formales	21
7.1.2.	Operaciones básicas de wolfie	21
7.1.3.	Operaciones básicas de itTítulos	22
7.2.	Representación	22
7.2.1.	Representación de wolfie	22
7.2.2.	Invariante de representación	23
7.2.3.	Función de abstracción	24
7.2.4.	Representación de itTítulos	25
7.2.5.	Invariante de Representación de itTítulos	25
7.3.	Algoritmos	25
7.3.1.	Algoritmos de wolfie	25
7.3.2.	Algoritmos de itTítulos	27
7.3.3.	Funciones auxiliares	27
7.4.	Servicios Usados	28

## 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:**  $O(1)$   
**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:**  $O(1)$   
**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:**  $O(T \cdot C \cdot |max\_nt|)$   
**Descripcion:** Devuelve un iterador a las promesas de un wolfie

**ACCIONESPORCLIENTE**(in  $c$ : cliente, in  $nt$ : nombre, 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:**  $O(\log(C) + |nt|)$   
**Descripcion:** Devuelve la cantidad de acciones que un cliente posee de un determinado título.

**INAUGURARWOLFIE**(in  $cs$ : conj(cliente))  $\rightarrow res$ : wolfie  
**Pre**  $\equiv \{\neg?(cs)\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{inaugurarWolfie}(cs)\}$   
**Complejidad:**  $O(\#(cs)^2)$   
**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:**  $O(|\text{nombre}(t)| + C)$

**ACTUALIZARCOTIZACIÓN**(in  $nt$ : nombre, in  $cot$ : nat, in/out  $w$ : wolfie)  
**Pre**  $\equiv \{w_0 =_{\text{obs}} w \wedge (\exists t:\text{título}) (t \in \text{títulos}(w) \wedge \text{nombre}(t) = nt)\}$

**Post**  $\equiv \{w =_{\text{obs}} \text{actualizarCotización}(nt, cot, w_0)\}$

**Complejidad:**  $O(C \cdot |nt| + C \cdot \log(C))$

**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:**  $O(|\text{título}(p)| + \log(C))$

**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:**  $O(|nt|)$

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

### 2.1.3. Operaciones básicas de itTítulos

CREARIT(**in**  $w$ : wolfie)  $\rightarrow res$ : itTítulo

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

**Post**  $\equiv \{res =_{\text{obs}} \text{crearItUni}(\text{títulos}(w))\}$

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

**Descripción:** Devuelve un iterador unidireccional a los títulos de wolfie.

ACTUAL(**in**  $i$ : itTítulo)  $\rightarrow res$ : título

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

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

**Complejidad:**  $O(|\text{título}(\text{actual}(i))|)$

**Descripción:** Devuelve el título actual.

PRÓXIMO(**in/out**  $i$ : itTítulo)  $\rightarrow res$ : [

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

**Post**  $\equiv \{H\} \text{ayPróximo}(i) \wedge i_0 = i \mid i =_{\text{obs}} \text{avanzar}(i_0)[O(1)] \text{ [Avanza el iterador.]}$

HAYPRÓXIMO(**in/out**  $i$ : itTítulo)  $\rightarrow res$ : bool

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

**Post**  $\equiv \{res =_{\text{obs}} \text{hayMas}(i)\}$

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

**Descripción:** Pregunta si hay más elementos para iterar.

## 2.2. Representación

### 2.2.1. Representación de wolfie

wolfie se representa con estr

donde estr es  $\text{tupla}(\text{títulos: diccString}(\text{infoTítulo}),$   
 $\text{clientes: conjEstNat}(\text{cliente})$   
 $\text{últimoLlamado: } \langle \text{cliente: cliente, promesas: conj(promesa), fueÚltimo: bool} \rangle)$   
 donde infoTítulo es  $\text{tupla}(\text{arrayClientes: array\_dimensionable}(\text{tuplaPorCliente}), \text{título: título, accDisponi-}$   
 $\text{bles: nat})$   
 donde tuplaPorCliente es  $\text{tupla}(\text{cliente: cliente, cantAcc: nat, promCompra: *promesa, promVenta: *promesa})$   
 Con un orden definido por  $a < b \Leftrightarrow a.\text{cliente} < b.\text{cliente}$   
 donde tuplaPorCantAcc es  $\text{tupla}(\text{cliente: cliente, cantAcc: nat, promCompra: *promesa, promVenta: *promesa})$   
 Con un orden definido por  $a < b \Leftrightarrow a.\text{cantAcc} > b.\text{cantAcc}$

### 2.2.2. Invariante de representación

- (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 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*.
- (VIII) Los títulos en infoTítulo tienen el mismo nombre que la clave que lleva a ellos.

Rep : estr  $\longrightarrow$  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)  $(\forall p: *promesa, t: \text{nombre}, c: \text{cliente}) \left( (p \neq \text{NULL} \wedge \text{def?}(t, e.\text{titulos}) \wedge_{\text{L}} \text{estáCliente?}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}) \wedge_{\text{L}} \text{buscarCliente}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).\text{promCompra}=p) \Rightarrow_{\text{L}} \text{título}(*p)=t \wedge \text{tipo}(*p)=\text{compra} \wedge (\text{límite}(*p) > \text{cotización}(\text{obtener}(t, e.\text{titulos}).\text{título}) \vee \text{cantidad}(*p) > \text{obtener}(t, e.\text{titulos}).\text{accDisponibles}) \right) \wedge$   
 (III)  $(\forall p: *promesa, t: \text{nombre}, c: \text{cliente}) \left( (p \neq \text{NULL} \wedge \text{def?}(t, e.\text{titulos}) \wedge_{\text{L}} \text{estáCliente?}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}) \wedge_{\text{L}} \text{buscarCliente}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).\text{promVenta}=p) \Rightarrow_{\text{L}} (\text{título}(*p)=t \wedge \text{tipo}(*p)=\text{venta} \wedge \text{límite}(*p) < \text{cotización}(\text{obtener}(t, e.\text{titulos}).\text{título})) \right) \wedge$   
 (IV)  $(\forall nt: \text{nombreT}) \left( \text{def?}(nt, e.\text{titulos}) \Rightarrow_{\text{L}} (\text{obtener}(nt, e.\text{titulos}).\text{accDisponibles} = \text{máximo}(\text{obtener}(nt, e.\text{titulos}).\text{título}) - \text{sumaAccClientes}(\text{obtener}(nt, e.\text{titulos}).\text{arrayClientes}, 0) \wedge \text{obtener}(nt, e.\text{titulos}).\text{accDisponibles} \geq 0) \right) \wedge$   
 (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**  $\left. \right) \wedge$   
 (VII)  $(\forall t: \text{nombre}) \text{def?}(t, e.\text{titulos}) \Rightarrow_{\text{L}} ((\forall i: \text{nat}) i < \text{longitud}(\text{buscar}(t, e.\text{titulos}).\text{arrayClientes})-1 \Rightarrow (\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}[i] < (\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}[i+1])))$   
 (VIII)  $(\forall t: \text{nombre}) \text{def?}(t, e.\text{titulos}) \Rightarrow_{\text{L}} t = \text{nombre}(\text{obtener}(t, e.\text{titulos}).\text{título})$   
  
 $\text{estáCliente?} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \longrightarrow \text{bool}$   
 $\text{estáCliente?}(c, a) \equiv \text{auxEstáCliente}(c, a, 0)$   
  
 $\text{auxEstáCliente} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{bool}$   
 $\text{auxEstáCliente}(c, a, i) \equiv \text{if } i=\text{longitud}(a) \text{ then false else } a[i].cliente = c \vee \text{auxEstáCliente}(c, a, i+1) \text{ fi}$   
  
 $\text{buscarCliente} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \longrightarrow \text{tuplaPorCliente} \quad \{\text{estáCliente}(c, a)\}$   
 $\text{buscarCliente}(c, a) \equiv \text{auxBuscarCliente}(c, a, 0)$   
  
 $\text{auxBuscarCliente} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{tuplaPorCliente} \quad \{\text{estáCliente}(c, a)\}$   
 $\text{auxBuscarCliente}(c, a, i) \equiv \text{if } a[i].cliente = c \text{ then } a[i] \text{ else } \text{auxBuscarCliente}(c, a, i+1) \text{ fi}$   
 $\text{sumaAccClientes} : \text{array\_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{nat}$   
 $\text{auxBuscarCliente}(a, i) \equiv \text{if } i=\text{longitud}(a) \text{ then } 0 \text{ else } a[i].cantAcc + \text{sumaAccClientes}(a, i+1) \text{ fi}$

### 2.2.3. Función de abstracción

$\text{Abs} : \text{estr } e \longrightarrow \text{wolfie} \quad \{\text{Rep}(e)\}$   
 $\text{Abs}(e) =_{\text{obs}} w: \text{wolfie} \mid \text{clientes}(w) = e.\text{clientes} \wedge (\forall t: \text{título}) (t \in \text{títulos}(w) \iff (\text{def?}(t, e.\text{titulos}) \wedge_{\text{L}} t = \text{obtener}(t, e.\text{titulos}).\text{título}))$   
 $\quad \wedge (\forall c: \text{cliente}) \text{promesasDe}(c, w) = \text{damePromesas}(\text{crearIt}(e.\text{titulos}), e, c) \wedge$   
 $\quad \text{accionesPorCliente}(c, t, w) = \text{buscarCliente}(\text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).cantAcc$   
 $\text{damePromesas} : \text{itDicc}(\text{diccString}(\text{infoT\~{A}tulo})) \times \text{estr} \times \text{cliente} \longrightarrow \text{conj}(\text{promesa})$

```

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

```

#### 2.2.4. Representación de itTítulos

itTítulos se representa con iterador

donde iterador es tupla(*it*: itClaves(infoTítulo),  
dicc: \*diccString(infoTítulo) )

#### 2.2.5. Invariante de Representación de itTítulos

Los el iterador de claves es iterador del diccString.

Rep : iterador  $\rightarrow$  bool

Rep(*i*)  $\equiv$  true  $\iff$  esIterador(i.it, CrearIt(\*(i.dicc)))

esIterador : itClaves(infoTítulo)  $\times$  itClaves(infoTítulo)  $\rightarrow$  bool

esIterador(it1, it2  $\equiv$  actual(it1)=actual(it2)  $\vee$  (hayMas?(it2)  $\wedge$   $\text{esIterador}(it1, \text{Avanzar}(it2))$ ))

Abs : iterador *i*  $\rightarrow$  itTítulo {Rep(*i*)}

Abs(*i*) =<sub>obs</sub> *t*: itTítulo | actual(*t*)=obtener(actual(i.it), \*(i.dicc))  $\wedge$  (hayMás(i.it) $\Rightarrow$ <sub>L</sub>(hayMás(*t*)  $\wedge$   $\text{Abs}(<\text{Avanzar}(i.it), i.dicc>, \text{avanzar}(t))$ ))

### 2.3. Algoritmos

#### 2.3.1. Algoritmos de wolfie

iClientes(in e: estr)  $\rightarrow$  res: itConjEstNat

```
1 return ( CrearIt ( e . clientes ) )
```

**Complejidad:** O(1)

iPromesasDe(in c: cliente, in/out e: estr)  $\rightarrow$  res: itConj(promesa)

```

1 if  $\neg$ (e.ultimoLlamado.cliente = c  $\wedge$  e.ultimoLlamado.fueUltimo) then O(1)
2   itClaves(diccString(infoTítulo)) it  $\leftarrow$  CrearIt ( e . titulos ) O(1)
3   conj(promesa) proms  $\leftarrow$  vacio() O(1)
4   tuplaPorClientes tup O(1)
5   while (HayMas?( it )) T* O(1)
6     tup  $\leftarrow$  BuscarCliente(Obtener(Nombre(Actual(it)), e . titulos).arrayClientes) O(C*|nombre(actual(it))|)
7     O(C*|nombre(actual(it))|)
8     if tup.promVenta  $\neq$  NULL then AgregarRapido(proms, *(tup.promVenta)) O(1)
9     if tup.promCompra  $\neq$  NULL then AgregarRapido(proms, *(tup.promCompra)) O(1)
10    Avanzar(it) O(1)
11  endWhile
12  e.ultimoLlamado.promesas  $\leftarrow$  proms O(1)
13 fi
14 return ( crearIt ( e . ultimoLlamado . promesas ) ) O(1)

```

**Complejidad:**  $4*O(1)+T*(O(1)+O(C*|nombre(actual(it))|))+3*O(1))+O(1)+O(1)\subseteq O(T*C*|max\_nt|)$

iAccionesPorCliente(in c: cliente, in nt, nombreT, in e: estr)  $\rightarrow$  res: nat



```
1  return (BuscarCliente(c, Obtener(nt, e.titulos)).cantAcc)
```

**Complejidad:**  $O(\log(C)+|nt|)$

iInaugurarWolfie(in c: conj(cliente))  $\rightarrow$  res: estr

```
1  res.titulos  $\leftarrow$  CrearDicc() O(1)
2  res.clientes  $\leftarrow$  NuevoConjEstNat(c) O(C(\log(C)))
3  res.ultimoLlamado  $\leftarrow$  <0, Vacio(), false> O(1)
```

**Complejidad:**  $O(C(\log(C)))$

iAgregarTítulo(in t: título, in/out e: estr)  $\rightarrow$  res: nat

```
1  Definir(e.titulos, nombre(t), <CrearArrayClientes(CrearIt(e.clientes), cardinal
2      (e.clientes)), t, #maxAcciones(t))
```

**Complejidad:**  $O(|nombre(t)|+C)$

iActualizarCotización(in nt: nombre, in cot: nat, in/out e: estr)

```
1  infoTítulo s  $\leftarrow$  Obtener(nt, e.titulos) O(|nt|)
2  recotizar(cot, s.titulo)
3  nat i  $\leftarrow$  0 O(1)
4  while i < |s.arrayClientes| C*
5      if (s.arrayClientes[i].promVenta  $\neq$  NULL \yluego limite(*(s.arrayClientes[i].promVenta))
6 > cotizacion(s.titulo)) then C*
7          s.arrayClientes[i].cantAcc -= cantidad(*(s.arrayClientes[i].promVenta)) O(1)
8          s.accDisponibles += cantidad(*(s.arrayClientes[i].promVenta)) O(1)
9          s.arrayClientes[i].promVenta = NULL O(1)
10     fi
11 endWhile
12 arreglo_dimensionable(tuplaPorCantAcc)[s.arrayClientes] arr O(C)
13 CambiarPorCantAcc(s.arrayClientes, arr) O(C)
14 heapsort(arr) O(C(\log(C)))
15 i  $\leftarrow$  0 O(1)
16 while i < |s.arrayClientes| C*
17     if (arr[i].promCompra  $\neq$  NULL \yluego limite(*(arr[i].promCompra)) < cotizacion(s.titulo)
18 cantidad(*(arr[i].promCompra))  $\leq$  s.accDisponibles) then O(1)
19         arr[i].cantAcc += cantidad(*(arr[i].promCompra)) O(1)
20         s.accDisponibles -= cantidad(*(arr[i].promCompra)) O(1)
21         arr[i].promCompra = NULL O(1)
22     fi
23     i++ O(1)
24 endWhile
25 CambiarPorCliente(arr, s.arrayClientes) O(C)
26 heapsort(s.arrayClientes) O(C(\log(C)))
```

**Complejidad:**  $O(|nt|)+2*O(1)+C*4*O(1)+O(C)+O(C)+O(C(\log(C)))+O(1)+C*4*O(1)+O(C)+O(C(\log(C)))=$   
 $O(|nt|+C(\log(C)))$

iAgregarPromesa(in c: cliente, in p:promesa, in/out e:estr)

```
1  promesa prom  $\leftarrow$  p O(1)
2  if tipo(prom)=compra then O(1)
3      BuscarCliente(c, Obtener(titulo(prom), e.titulos).arrayClientes).promCompra  $\leftarrow$  &prom
4      O(|titulo(p)|+C)
5  else
6      BuscarCliente(c, Obtener(titulo(prom), e.titulos).arrayClientes).promCompra  $\leftarrow$  &prom
7      O(|titulo(p)|+C)
8  fi
```

**Complejidad:**  $O(1)+O(1)+O(|titulo(p)|+C)=O(|titulo(p)|+C)$

iEnAlza(in nt: nombreT, in e: estr)  $\rightarrow$  res: bool

```
1  return (enAlza(Obtener(nt, e.titulos).titulo))
```

**Complejidad:**  $O(|nt|)$

**2.3.2. Algoritmos de itTítulos**

iCrearIt(in e: estr) → res: iterador

```
1  return(<crearIt(e.titulos), &(e.titulos)>)
```

**Complejidad:**  $O(|nt|)$

iActual(in i: iterador) → res: titulo

```
1  return(Significado(Actual(i.it), *(i.dicc)).titulo)
```

**Complejidad:**  $O(|nt|)$

iPróximo(in/out i: iterador)

```
1  avanzar(i.it)
```

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

iHayPróximo(in i: iterador) → res: bool

```
1  return(HayMas(i.it))
```

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

**2.3.3. Funciones auxiliares**

CrearArrayClientes(in it: itConjEstNat, in n: nat) → res: arreglo\_dimensionable(tuplaPorClientes)

```
1  arreglo_dimensionable(tuplaPorClientes)[n] arr  O(n)
2  nat i ← 0
3  do
4      arr[i] ← <Actual(it), 0, NULL, NULL>
5      i++
6      Proximo(it)
7  while hayProx(it)
8  return arr
```

n\*  
O(1)  
O(1)  
O(1)  
O(1)

**Complejidad:**  $O(n)+O(1)+n*4*O(1)=O(n)$

CambiarPorCantAcc(in a1: arreglo\_dimensionable(tuplaPorCliente), in/out a2: arreglo\_dimensionable(tuplaPorCantAcc))

```
1  nat i ← 0
2  while i < |a1|
3      a2[i].cliente ← a1[i].cliente
4      a2[i].cantAcc ← a1[i].cantAcc
5      a2[i].promCompra ← a1[i].promCompra
6      a2[i].promVenta ← a1[i].promVenta
7      i++
8  end While
```

O(1)  
|a1|\*  
O(1)  
O(1)  
O(1)  
O(1)  
O(1)

**Complejidad:**  $O(1)+|a1|*5*O(1)=O(|a1|)$

CambiarPorCliente(in a1: arreglo\_dimensionable(tuplaPorCantAcc), in/out a2: arreglo\_dimensionable(tuplaPorCliente))

```
1  nat i ← 0
2  while i < |a1|
3      a2[i].cliente ← a1[i].cliente
4      a2[i].cantAcc ← a1[i].cantAcc
5      a2[i].promCompra ← a1[i].promCompra
6      a2[i].promVenta ← a1[i].promVenta
7      i++
8  end While
```

O(1)  
|a1|\*  
O(1)  
O(1)  
O(1)  
O(1)  
O(1)

**Complejidad:**  $O(1)+|a1|*5*O(1)=O(|a1|)$

BUSCARCLIENTE(in cliente: cliente, in a: arreglo\_dimensionable(tuplaPorCliente)) → res = tuplaPorCliente

```

1  int: arriba ← longitud(a)
2  int: abajo ← 0
3  int: centro
4  while (abajo ≤ arriba)
5      centro ← (arriba + abajo)/2;
6      if (arreglo[centro].Π1 == cliente)
7          return a[centro];
8      else
9          if (cliente < arreglo[centro].Π1)
10             arriba ← centro-1;
11          else
12             abajo ← centro+1;
13          endIf
14      endIf
15  endWhile

```

**Complejidad**  $O(\log(|a|))$  porque es una implementación del algoritmo de búsqueda, que por lo visto en clase, tiene complejidad logarítmica en la longitud del arreglo.

## 2.4. Servicios Usados

Módulo	Operación	Complejidad Requerida
diccString(infoTitulo)	CrearIt	$O(1)$
diccString(infoTitulo)	Definir	$ nt $
diccString(infoTitulo)	Obtener	$ nt $
conj(promesa)	Vacio	$O(1)$
conj(promesa)	AgregarRapido	$O(1)$
itDicc(diccString(infoTitulo))	HayMás	$O(1)$
itDicc(diccString(infoTitulo))	Actual	$O(1)$
itDicc(diccString(infoTitulo))	Avanzar	$O(1)$
	BuscarCliente	$O(\log(C))$
conjEstNat	NuevoConjEstNat	$O(C(\log(C)))$
itConjEstNat	CrearIt	$O(1)$
itConjEstNat	HayProx	$O(1)$
itConjEstNat	Proximo	$O(1)$
itConjEstNat	Actual	$O(1)$
arreglo_dimensionable	CrearNuevo	$O(n)$
arreglo_dimensionable	AgregarElemento	$O(1)$
arreglo_dimensionable	•[•]	$O(1)$
	heapsort	$O(n(\log(n)))$

## 3. Módulo Diccionario String(alpha)

### 3.1. Interfaz

#### 3.1.1. Parámetros formales

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

**géneros:** diccString( $\alpha$ ), itClaves(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**  $d$ : diccString( $\alpha$ ), **in**  $c$ : 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**  $d$ : diccString( $\alpha$ ), **in**  $c$ : string)  $\rightarrow res$ : 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**  $d$ : diccString( $\alpha$ ), **in**  $c$ : 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**  $d$ : diccString( $\alpha$ ))  $\rightarrow res$ : itClaves(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 de Diccionario String.

**HAYMAS?**(**in**  $d$ : itClaves(string))  $\rightarrow res$ : bool

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

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

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

**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:**  $O(\text{longitud}(\text{secuSuby}(d)))$

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

**AVANZAR**(**in/out**  $it$ : itClaves(string))  $\rightarrow res$ : [

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

**Post**  $\equiv \{hayMas?(it) \wedge it = it_0 \mid it =_{\text{obs}} \text{avanzar}(it_0) [O(\text{longitud}(\text{secuSuby}(d)))] [\text{Avanza a la próxima clave.}]$

## 3.2. Representación

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

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

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

Nodo se representa con  $\text{estrNodo}$

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

### 3.2.2. Invariante de Representación de $\text{diccString}$

- (I) Existe un único camino entre cada nodo y el nodo raíz (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) Raíz es distinto de NULL
- (IV) En claves está el camino que se recorre desde la raíz hasta cada nodo hoja.

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

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

### 3.2.3. Operaciones auxiliares del invariante de Representación

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

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

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

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

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

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

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

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

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

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

$\text{todasLasHojas} : \text{puntero(nodo)} \times \text{nat} \longrightarrow \text{conj(nodo)}$

$\text{todasLasHojas}(p, n) \equiv \text{if } p == \text{NULL} \text{ then}$   
 $\quad \text{false}$   
 $\text{else}$   
 $\quad \text{if esHoja}(p) \text{ then Ag}(*p, \text{vacío}) \text{ else auxTodasLasHojas}((p).\text{hijos}, 256) \text{ fi}$   
 $\text{fi}$

```

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

```

### 3.2.4. Función de abstracción de diccString

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

### 3.2.5. 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 = estrDicc(α)
1 n ← nodo                                O(1)
2 n ← crearNodo()                          O(1)
3 raiz ← *n                                O(1)

```

**Complejidad:**  $3 \cdot O(1) = O(1)$

```

ICREARNODO() → res = nodo
1 d : arreglo_estatico[256]                O(1)
2 i ← 0                                    O(1)
3 while (i < 256)                           256*
4     d[i] ← NULL                           O(1)
5 endwhile
6 hijos ← d                                O(1)
7 valor ← NULL                             O(1)

```

**Complejidad:**  $2 \cdot O(1) + 256 \cdot O(1) + 2 \cdot O(1) = O(1)$

```

IDEFINIR(in/out estrDicc(α): d, in string: c, in alfa: s)
1 i ← 0                                    O(1)
2 p ← d.raiz                              O(1)
3 while (i < (longitud(s)))                 |s|*
4     if (p.hijos[ord(s[i])] == NULL)        O(1)
5         n: nodo ← crearNodo()              O(1)
6         p.hijos[ord(s[i])] ← *n            O(1)

```

```

7         endIf
8 p ← p.hijos[ord(s[i])]
9 i++
10 endWhile
11 p.valor ← a
12 agregarAdelante(hijos, c)

```

O(1)  
O(1)  
O(1)  
O(|s|)

**Complejidad:**  $2*O(1) + |s|*5*O(1) + O(1) + O(|s|) = O(|s|)$

ISIGNIFICADO(in estrDicc( $\alpha$ ): d, in string: c) → 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

```

O(1)  
O(1)  
 $|s|*$   
O(1)  
O(1)  
O(1)

**Complejidad:**  $2*O(1) + |s|*2*O(1) + O(1) = O(|s|)$

IDEFINIDO?(in estrDicc( $\alpha$ ): 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++ O(1)
7     else
8         return false
9     endIf
10 endWhile
11 return p.valor != NULL

```

O(1)  
O(1)  
 $|s|*$   
O(1)  
O(1)  
O(1)  
O(1)

**Complejidad:**  $2*O(1) + |s|*3*O(1) + O(1) = O(|s|)$

ICLAVES(in estrDicc( $\alpha$ ): d) → res = lista\_enlazada(string)

```

1 return itClaves(d)

```

O(1)

**Complejidad:** O(1)

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

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

### 3.4. Servicios Usados

Módulo	Operación	Complejidad Requerida
arreglo_estático	AgregarElemento	O(1)
arreglo_estático	•[•]	O(1)
lista	AgregarAdelante	O(copy( $\alpha$ ))
lista	•[•]	O(1)

## 4. Módulo Conjunto Estático de Nats

### 4.1. Interfaz

**géneros** conjEstNat, itConjEstNat

Se explica con: CONJUNTO(NAT), ITERADOR UNIDIRECCIONAL(NAT). Usa:

#### 4.1.1. Operaciones básicas de conjEstNat

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

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

**CARDINAL**(in  $c$ : conjEstNat)  $\rightarrow res$ : nat  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \#c\}$   
**Complejidad:**  $O(n)$   
**Descripcion:** Devuelve la cantidad de elementos que hay en el conjunto

#### 4.1.2. Operaciones básicas de itConjEstNat

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

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

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

**HAYPRÓX?**(in  $i$ : itConjEstNat)  $\rightarrow res$ : bool  
**Pre**  $\equiv \{i_0 = i\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{hayMas?}(i)\}$   
**Complejidad:**  $O(1)$   
**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  $\iff (\forall i: \text{nat}) (i < \text{longitud}(a)-1 \Rightarrow (\text{definido?}(a, i) \wedge \text{definido?}(a, i+1) \wedge a[i] < a[i+1]))$

### 4.2.2. Función de abstracción de conjEstNat

Abs : array  $a \rightarrow$  conjEstNat

{Rep( $a$ )}

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

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

### 4.2.3. 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  $\iff i.pos < \text{longitud}(*i.lista)$

### 4.2.4. Función de abstracción de itConjEstNat

Abs : iterador  $it \rightarrow$  itConjEstNat

{Rep( $it$ )}

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

## 4.3. Algoritmos

### 4.3.1. Algoritmos de conjEstNat

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

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

**Complejidad:**  $O(1)+O(n)+O(1)+n*(O(1)+O(1)+O(1))+O(n(\log(n))) = O(n(\log(n)))$

**Aclaraciones:** Utilizamos el algoritmo HEAPSORT provisto en el apunte ALGORITMOS BÁSICOS, con las complejidades allí descriptas.

iPertenece(in n: nat, in c: array) → res: bool

1	<b>bool</b> b ← <b>false</b>	O(1)
2	<b>nat</b> i ← 0	O(1)
3	<b>while</b> (i <  c )	
4	b ← (b ∨ c[i]=n)	n*
5	i++	O(1)
6	<b>endWhile</b>	O(1)
7	<b>return</b> (b)	

**Complejidad:**  $O(1)+O(1)+n*(O(1)+O(1)) = O(n)$

#### 4.3.2. Algoritmos de itConjEstNat

iCrearIt(in a: array) → res: iterador

1 **return** (<0, &a>)

**Complejidad:** O(1)

iActual(in it: iterador) → res: nat

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

**Complejidad:** O(1)

iActual(in/out it: iterador)

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

**Complejidad:** O(1)

iHayPróximo?(in it: iterador) → res: bool

1 **return** (it.pos+1 < longitud(it.lista))

**Complejidad:** O(1)

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

#### 4.4. Servicios Usados

Módulo	Operación	Complejidad Requerida
arreglo_estático	CrearNuevo	O(n)
arreglo_estático	AgregarElemento	O(1)
arreglo_estático	•[•]	O(1)
	heapsort	O(n(log(n)))

#### 4.5. TAD CONJUNTO ESTÁTICO DE NATS

TAD CONJUNTO ESTÁTICO DE NATS

igualdad observacional

$$(\forall c, c' : \text{conjEstNat}) (c =_{\text{obs}} c' \iff ((\forall a : \text{nat})(a \in c =_{\text{obs}} a \in c')))$$

**géneros**       $\text{conjEstNat}$

**exporta**       $\text{conjEstNat}, \text{generadores}, \text{observadores}, \#$

**usa**             $\text{BOOL}, \text{NAT}, \text{CONJUNTO}(\text{NAT})$

**observadores básicos**

$\bullet \in \bullet$             :  $\text{nat} \times \text{conjEstNat}$              $\longrightarrow \text{bool}$

**generadores**

$\text{crearConjEstNat} : \text{conj}(\text{nat})$              $\longrightarrow \text{conj}(\text{EstNat})$

**otras operaciones**

$\#$                 :  $\text{conj}(\text{EstNat})$              $\longrightarrow \text{nat}$

**axiomas**       $\forall c : \text{conj}(\text{nat}), \forall n : \text{nat}$

$n \in \text{crearConjEstNat}(c) \equiv (n \in c)$

$\#(\text{crearConjEstNat}(c)) \equiv \#(c)$

**Fin TAD**

## 5. Módulo Promesa

### 5.1. Interfaz

#### 5.1.1. Parámetros formales

**géneros**       $\text{promesa}$

**se explica con:** PROMESA.

#### 5.1.2. Operaciones básicas de promesa

**TÍTULO**( $\text{in } p : \text{promesa}$ )  $\rightarrow \text{res} : \text{nombre}$

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

**Post**  $\equiv \{\text{res} =_{\text{obs}} \text{titulo}(p)\}$

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

**Descripcion:** Devuelve el nombre del título de la promesa.

**TIPO**( $\text{in } p : \text{promesa}$ )  $\rightarrow \text{res} : \text{tipoPromesa}$

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

**Post**  $\equiv \{\text{res} =_{\text{obs}} \text{tipo}(p)\}$

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

**Descripcion:** Devuelve el tipo de promesa de la promesa.

**LIMITE**( $\text{in } p : \text{promesa}$ )  $\rightarrow \text{res} : \text{dinero}$

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

**Post**  $\equiv \{\text{res} =_{\text{obs}} \text{limite}(p)\}$

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

**Descripcion:** Devuelve el limite de la promesa.

**CANTIDAD**(in  $p$ : promesa)  $\rightarrow res$  : 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)  $\rightarrow res$  : promesa  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{crearPromesa}(t, tipo, n, m)\}$   
**Complejidad:**  $O(|t|)$   
**Descripcion:** Devuelve una nueva promesa.

## 5.2. Representación

### 5.2.1. Representación de promesa

promesa se representa con estr

donde estr es tupla(*título*: nombre *tipo*: tipoPromesa *limite*: dinero *cantidad*: nat )

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

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

### 5.2.2. Abs

$\text{Abs} : \text{estr } e \rightarrow \text{promesa} \quad \{\text{Rep}(e)\}$

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

## 5.3. Algoritmos

### 5.3.1. Algoritmos de promesa

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

1  $res \leftarrow e.t \setminus 'ititulo$

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

1  $res \leftarrow e.tipo$

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

1  $res \leftarrow e.limite$

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

1  $res \leftarrow e.cantidad$

iCrearPromesa(in nombre: t, in TipoPromes: tipo, in dinero: n, in nat: m)  $\rightarrow res = \text{Promesa}$

1  $res.t \setminus 'ititulo \leftarrow t$

2  $res.tipo \leftarrow tipo$

3  $res.limite \leftarrow n$

4  $res.cantidad \leftarrow m$

## 6. Módulo Título

### 6.1. Interfaz

#### 6.1.1. Parámetros formales

**géneros**    **título**

**se explica con:** TÍTULO.

#### 6.1.2. Operaciones básicas de título

**NOMBRE**(**in**  $t$ : título)  $\rightarrow res$  : nombre  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{nombre}(t)\}$   
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve el nombre del título

**#MÁXACCIONES**(**in**  $t$ : título)  $\rightarrow res$  : nat  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \#máxAcciones(t)\}$   
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve el máximo de cantidad de acciones.

**COTIZACIÓN**(**in**  $t$ : título)  $\rightarrow res$  : dinero  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{cotizacion}(t)\}$   
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve la cotización del título.

**CREARTÍTULO**(**in**  $t$ : nombre, **in**  $c$ : dinero, **in**  $n$ : nat)  $\rightarrow res$  : título  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{crearTítulo}(t, c, n)\}$   
**Complejidad:**  $O(|t|)$   
**Descripcion:** Devuelve una nueva promesa.

**RECOTIZAR**(**in**  $d$ : dinero, **in**  $t$ : título)  $\rightarrow res$  : título  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{recotizar}(d, t)\}$   
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve una nueva promesa.

## 7. Módulo Wolfie

### 7.1. Interfaz

#### 7.1.1. Parámetros formales

**géneros**    wolfie

**se explica con:** WOLFIE.

#### 7.1.2. 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:**  $O(1)$   
**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:**  $O(1)$   
**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:**  $O(T \cdot C \cdot |max\_nt|)$   
**Descripcion:** Devuelve un iterador a las promesas de un wolfie

**ACCIONESPORCLIENTE**(**in**  $c$ : cliente, **in**  $nt$ : nombre, **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:**  $O(\log(C) + |nt|)$   
**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:**  $O(\#(cs)^2)$   
**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:**  $O(|\text{nombre}(t)| + C)$

**ACTUALIZARCOTIZACIÓN**(**in**  $nt$ : nombre, **in**  $cot$ : nat, **in/out**  $w$ : wolfie)  
**Pre**  $\equiv \{w_0 =_{\text{obs}} w \wedge (\exists t:\text{título}) (t \in \text{títulos}(w) \wedge \text{nombre}(t) = nt)\}$   
**Post**  $\equiv \{w =_{\text{obs}} \text{actualizarCotización}(nt, cot, w_0)\}$   
**Complejidad:**  $O(C \cdot |nt| + C \cdot \log(C))$   
**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)  
**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:**  $O(|\text{título}(p)| + \log(C))$

**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:**  $O(|nt|)$

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

### 7.1.3. Operaciones básicas de itTítulos

**CREARIT**(**in**  $w$ : wolfie)  $\rightarrow res$ : itTítulo

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

**Post**  $\equiv \{res =_{\text{obs}} \text{crearItUni}(\text{títulos}(w))\}$

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

**Descripción:** Devuelve un iterador unidireccional a los títulos de wolfie.

**ACTUAL**(**in**  $i$ : itTítulo)  $\rightarrow res$ : título

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

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

**Complejidad:**  $O(|\text{título}(\text{actual}(i))|)$

**Descripción:** Devuelve el título actual.

**PRÓXIMO**(**in/out**  $i$ : itTítulo)  $\rightarrow res$ : [

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

**Post**  $\equiv \{H\} \text{ayPróximo}(i) \wedge i_0 = i \mid i =_{\text{obs}} \text{avanzar}(i_0)[O(1)] \text{ [Avanza el iterador.]}$

**HAYPRÓXIMO**(**in/out**  $i$ : itTítulo)  $\rightarrow res$ : bool

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

**Post**  $\equiv \{res =_{\text{obs}} \text{hayMas}(i)\}$

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

**Descripción:** Pregunta si hay más elementos para iterar.

## 7.2. Representación

### 7.2.1. Representación de wolfie

wolfie se representa con estr

donde **estr** es  $\text{tupla}(\text{títulos: diccString}(\text{infoTítulo}),$   
 $\text{clientes: conjEstNat}(\text{cliente})$   
 $\text{últimoLlamado: } \langle \text{cliente: cliente, promesas: conj}(\text{promesa}), \text{ fueUltimo: bool} \rangle)$

donde **infoTítulo** es  $\text{tupla}(\text{arrayClientes: array\_dimensionable}(\text{tuplaPorCliente}), \text{ título: título, accDisponi-}$   
 $\text{bles: nat})$

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

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

### 7.2.2. Invariante de representación

- (I) Los clientes de *clientes* son los mismos que hay dentro de *titulos*.
- (II) Las promesas de compra son de su título y cliente y no cumplen los requisitos para ejecutarse.
- (III) Las promesas de 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.titulos*.
- (VIII) Los títulos en *infoTítulo* tienen el mismo nombre que la clave que lleva a ellos.

Rep : `estr`  $\longrightarrow$  `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)  $(\forall p: *promesa, t: \text{nombre}, c: \text{cliente}) \left( (p \neq \text{NULL} \wedge \text{def?}(t, e.\text{titulos}) \wedge_{\text{L}} \text{estáCliente?}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}) \wedge_{\text{L}} \text{buscarCliente}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).\text{promCompra}=p) \Rightarrow_{\text{L}} \text{título}(*p)=t \wedge \text{tipo}(*p)=\text{compra} \wedge (\text{límite}(*p) > \text{cotización}(\text{obtener}(t, e.\text{titulos}).\text{título}) \vee \text{cantidad}(*p) > \text{obtener}(t, e.\text{titulos}).\text{accDisponibles}) \right) \wedge$   
 (III)  $(\forall p: *promesa, t: \text{nombre}, c: \text{cliente}) \left( (p \neq \text{NULL} \wedge \text{def?}(t, e.\text{titulos}) \wedge_{\text{L}} \text{estáCliente?}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}) \wedge_{\text{L}} \text{buscarCliente}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).\text{promVenta}=p) \Rightarrow_{\text{L}} (\text{título}(*p)=t \wedge \text{tipo}(*p)=\text{venta} \wedge \text{límite}(*p) < \text{cotización}(\text{obtener}(t, e.\text{titulos}).\text{título})) \right) \wedge$   
 (IV)  $(\forall nt: \text{nombreT}) \left( \text{def?}(nt, e.\text{titulos}) \Rightarrow_{\text{L}} (\text{obtener}(nt, e.\text{titulos}).\text{accDisponibles} = \text{máximo}(\text{obtener}(nt, e.\text{titulos}).\text{título}) - \text{sumaAccClientes}(\text{obtener}(nt, e.\text{titulos}).\text{arrayClientes}, 0) \wedge \text{obtener}(nt, e.\text{titulos}).\text{accDisponibles} \geq 0) \right) \wedge$   
 (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**  $\left. \right) \wedge$   
 (VII)  $(\forall t: \text{nombre}) \text{def?}(t, e.\text{titulos}) \Rightarrow_{\text{L}} ((\forall i: \text{nat}) i < \text{longitud}(\text{buscar}(t, e.\text{titulos}).\text{arrayClientes})-1 \Rightarrow (\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}[i] < (\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}[i+1])))$   
 (VIII)  $(\forall t: \text{nombre}) \text{def?}(t, e.\text{titulos}) \Rightarrow_{\text{L}} t = \text{nombre}(\text{obtener}(t, e.\text{titulos}).\text{título})$   
  
 $\text{estáCliente?} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \longrightarrow \text{bool}$   
 $\text{estáCliente?}(c, a) \equiv \text{auxEstáCliente}(c, a, 0)$   
  
 $\text{auxEstáCliente} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{bool}$   
 $\text{auxEstáCliente}(c, a, i) \equiv \text{if } i=\text{longitud}(a) \text{ then false else } a[i].cliente = c \vee \text{auxEstáCliente}(c, a, i+1) \text{ fi}$   
  
 $\text{buscarCliente} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \longrightarrow \text{tuplaPorCliente} \quad \{\text{estáCliente}(c, a)\}$   
 $\text{buscarCliente}(c, a) \equiv \text{auxBuscarCliente}(c, a, 0)$   
  
 $\text{auxBuscarCliente} : \text{cliente} \times \text{array\_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{tuplaPorCliente} \quad \{\text{estáCliente}(c, a)\}$   
 $\text{auxBuscarCliente}(c, a, i) \equiv \text{if } a[i].cliente = c \text{ then } a[i] \text{ else } \text{auxBuscarCliente}(c, a, i+1) \text{ fi}$   
 $\text{sumaAccClientes} : \text{array\_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{nat}$   
 $\text{auxBuscarCliente}(a, i) \equiv \text{if } i=\text{longitud}(a) \text{ then } 0 \text{ else } a[i].cantAcc + \text{sumaAccClientes}(a, i+1) \text{ fi}$

### 7.2.3. Función de abstracción

$\text{Abs} : \text{estr } e \longrightarrow \text{wolfie} \quad \{\text{Rep}(e)\}$   
 $\text{Abs}(e) =_{\text{obs}} w: \text{wolfie} \mid \text{clientes}(w) = e.\text{clientes} \wedge (\forall t: \text{título}) (t \in \text{títulos}(w) \iff (\text{def?}(t, e.\text{titulos}) \wedge_{\text{L}} t = \text{obtener}(t, e.\text{titulos}).\text{título}))$   
 $\quad \wedge (\forall c: \text{cliente}) \text{promesasDe}(c, w) = \text{damePromesas}(\text{crearIt}(e.\text{titulos}), e, c) \wedge$   
 $\quad \text{accionesPorCliente}(c, t, w) = \text{buscarCliente}(\text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).cantAcc$   
 $\text{damePromesas} : \text{itDicc}(\text{diccString}(\text{infoT\~{A}tulo})) \times \text{estr} \times \text{cliente} \longrightarrow \text{conj}(\text{promesa})$

```

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

```

#### 7.2.4. Representación de itTítulos

itTítulos se representa con iterador

donde iterador es tupla(*it*: itClaves(infoTítulo),  
dicc: \*diccString(infoTítulo) )

#### 7.2.5. Invariante de Representación de itTítulos

Los el iterador de claves es iterador del diccString.

Rep : iterador  $\rightarrow$  bool

Rep(*i*)  $\equiv$  true  $\iff$  esIterador(i.it, CrearIt(\*(i.dicc)))

esIterador : itClaves(infoTítulo)  $\times$  itClaves(infoTítulo)  $\rightarrow$  bool

esIterador(it1, it2  $\equiv$  actual(it1)=actual(it2)  $\vee$  (hayMas?(it2)  $\wedge$   $\text{esIterador}(it1, \text{Avanzar}(it2))$ ))

Abs : iterador *i*  $\rightarrow$  itTítulo {Rep(*i*)}

Abs(*i*) =<sub>obs</sub> *t*: itTítulo | actual(*t*)=obtener(actual(i.it), \*(i.dicc))  $\wedge$  (hayMás(i.it) $\Rightarrow$ <sub>L</sub>(hayMás(*t*)  $\wedge$   $\text{Abs}(<\text{Avanzar}(i.it), i.dicc>, \text{avanzar}(t))$ ))

### 7.3. Algoritmos

#### 7.3.1. Algoritmos de wolfie

iClientes(in e: estr)  $\rightarrow$  res: itConjEstNat

```
1 return ( CrearIt ( e . clientes ) )
```

**Complejidad:** O(1)

iPromesasDe(in c: cliente, in/out e: estr)  $\rightarrow$  res: itConj(promesa)

```

1 if  $\neg$ (e.ultimoLlamado.cliente = c  $\wedge$  e.ultimoLlamado.fueUltimo) then O(1)
2   itClaves(diccString(infoTítulo)) it  $\leftarrow$  CrearIt ( e . titulos ) O(1)
3   conj(promesa) proms  $\leftarrow$  vacio() O(1)
4   tuplaPorClientes tup O(1)
5   while (HayMas?(it)) T* O(1)
6     tup  $\leftarrow$  BuscarCliente(Obtener(Nombre(Actual(it)), e. titulos).arrayClientes) O(C*|nombre(actual(it))|)
7     O(C*|nombre(actual(it))|)
8     if tup.promVenta  $\neq$  NULL then AgregarRapido(proms, *(tup.promVenta)) O(1)
9     if tup.promCompra  $\neq$  NULL then AgregarRapido(proms, *(tup.promCompra)) O(1)
10    Avanzar(it) O(1)
11  endWhile
12  e.ultimoLlamado.promesas  $\leftarrow$  proms O(1)
13 fi
14 return ( crearIt ( e . ultimoLlamado . promesas ) ) O(1)

```

**Complejidad:**  $4*O(1)+T*(O(1)+O(C*|nombre(actual(it))|))+3*O(1))+O(1)+O(1)\subseteq O(T*C*|max\_nt|)$

iAccionesPorCliente(in c: cliente, in nt, nombreT, in e: estr)  $\rightarrow$  res: nat

```
1  return (BuscarCliente(c, Obtener(nt, e.titulos)).cantAcc)
```

**Complejidad:**  $O(\log(C)+|nt|)$

iInaugurarWolfie(in c: conj(cliente))  $\rightarrow$  res: estr

```
1  res.titulos  $\leftarrow$  CrearDicc() O(1)
2  res.clientes  $\leftarrow$  NuevoConjEstNat(c) O(C(\log(C)))
3  res.ultimoLlamado  $\leftarrow$  <0, Vacio(), false> O(1)
```

**Complejidad:**  $O(C(\log(C)))$

iAgregarTítulo(in t: título, in/out e: estr)  $\rightarrow$  res: nat

```
1  Definir(e.titulos, nombre(t), <CrearArrayClientes(CrearIt(e.clientes), cardinal
2      (e.clientes)), t, #maxAcciones(t))
```

**Complejidad:**  $O(|nombre(t)|+C)$

iActualizarCotización(in nt: nombre, in cot: nat, in/out e: estr)

```
1  infoTítulo s  $\leftarrow$  Obtener(nt, e.titulos) O(|nt|)
2  recotizar(cot, s.titulo)
3  nat i  $\leftarrow$  0 O(1)
4  while i < |s.arrayClientes| C*
5      if (s.arrayClientes[i].promVenta  $\neq$  NULL \yluego limite(*(s.arrayClientes[i].promVenta))
6 > cotizacion(s.titulo)) then O(1)
7          s.arrayClientes[i].cantAcc -= cantidad(*(s.arrayClientes[i].promVenta)) O(1)
8          s.accDisponibles += cantidad(*(s.arrayClientes[i].promVenta)) O(1)
9          s.arrayClientes[i].promVenta = NULL O(1)
10     fi
11 endWhile
12 arreglo_dimensionable(tuplaPorCantAcc)[s.arrayClientes] arr O(C)
13 CambiarPorCantAcc(s.arrayClientes, arr) O(C)
14 heapsort(arr) O(C(\log(C)))
15 i  $\leftarrow$  0 O(1)
16 while i < |s.arrayClientes| C*
17     if (arr[i].promCompra  $\neq$  NULL \yluego limite(*(arr[i].promCompra)) < cotizacion(s.titulo)
18 cantidad(*(arr[i].promCompra))  $\leq$  s.accDisponibles) then O(1)
19         arr[i].cantAcc += cantidad(*(arr[i].promCompra)) O(1)
20         s.accDisponibles -= cantidad(*(arr[i].promCompra)) O(1)
21         arr[i].promCompra = NULL O(1)
22     fi
23     i++ O(1)
24 endWhile
25 CambiarPorCliente(arr, s.arrayClientes) O(C)
26 heapsort(s.arrayClientes) O(C(\log(C)))
```

**Complejidad:**  $O(|nt|)+2*O(1)+C*4*O(1)+O(C)+O(C)+O(C(\log(C)))+O(1)+C*4*O(1)+O(C)+O(C(\log(C)))=$   
 $O(|nt|+C(\log(C)))$

iAgregarPromesa(in c: cliente, in p:promesa, in/out e:estr)

```
1  promesa prom  $\leftarrow$  p O(1)
2  if tipo(prom)=compra then O(1)
3      BuscarCliente(c, Obtener(titulo(prom), e.titulos).arrayClientes).promCompra  $\leftarrow$  &prom
4      O(|titulo(p)|+C)
5  else
6      BuscarCliente(c, Obtener(titulo(prom), e.titulos).arrayClientes).promCompra  $\leftarrow$  &prom
7      O(|titulo(p)|+C)
8  fi
```

**Complejidad:**  $O(1)+O(1)+O(|titulo(p)|+C)=O(|titulo(p)|+C)$

iEnAlza(in nt: nombreT, in e: estr)  $\rightarrow$  res: bool

```
1  return (enAlza(Obtener(nt, e.titulos).titulo))
```

**Complejidad:**  $O(|nt|)$

**7.3.2. Algoritmos de itTítulos**

iCrearIt(in e: estr) → res: iterador

```
1  return(<crearIt(e.titulos), &(e.titulos)>)
```

**Complejidad:**  $O(|nt|)$

iActual(in i: iterador) → res: titulo

```
1  return(Significado(Actual(i.it), *(i.dicc)).titulo)
```

**Complejidad:**  $O(|nt|)$

iPróximo(in/out i: iterador)

```
1  avanzar(i.it)
```

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

iHayPróximo(in i: iterador) → res: bool

```
1  return(HayMas(i.it))
```

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

**7.3.3. Funciones auxiliares**

CrearArrayClientes(in it: itConjEstNat, in n: nat) → res: arreglo\_dimensionable(tuplaPorClientes)

```
1  arreglo_dimensionable(tuplaPorClientes)[n] arr  O(n)
2  nat i ← 0
3  do
4      arr[i] ← <Actual(it), 0, NULL, NULL>
5      i++
6      Proximo(it)
7  while hayProx(it)
8  return arr
```

n\*  
O(1)  
O(1)  
O(1)  
O(1)

**Complejidad:**  $O(n)+O(1)+n*4*O(1)=O(n)$

CambiarPorCantAcc(in a1: arreglo\_dimensionable(tuplaPorCliente), in/out a2: arreglo\_dimensionable(tuplaPorCantAcc))

```
1  nat i ← 0
2  while i < |a1|
3      a2[i].cliente ← a1[i].cliente
4      a2[i].cantAcc ← a1[i].cantAcc
5      a2[i].promCompra ← a1[i].promCompra
6      a2[i].promVenta ← a1[i].promVenta
7      i++
8  end While
```

O(1)  
|a1|\*  
O(1)  
O(1)  
O(1)  
O(1)  
O(1)

**Complejidad:**  $O(1)+|a1|*5*O(1)=O(|a1|)$

CambiarPorCliente(in a1: arreglo\_dimensionable(tuplaPorCantAcc), in/out a2: arreglo\_dimensionable(tuplaPorCliente))

```
1  nat i ← 0
2  while i < |a1|
3      a2[i].cliente ← a1[i].cliente
4      a2[i].cantAcc ← a1[i].cantAcc
5      a2[i].promCompra ← a1[i].promCompra
6      a2[i].promVenta ← a1[i].promVenta
7      i++
8  end While
```

O(1)  
|a1|\*  
O(1)  
O(1)  
O(1)  
O(1)  
O(1)

**Complejidad:**  $O(1)+|a1|*5*O(1)=O(|a1|)$

BUSCARCLIENTE(in cliente: cliente, in a: arreglo\_dimensionable(tuplaPorCliente)) → res = tuplaPorCliente

```

1 int: arriba ← longitud(a)
2 int: abajo ← 0
3 int: centro
4 while (abajo ≤ arriba)
5     centro ← (arriba + abajo)/2;
6     if (arreglo[centro].Π1 == cliente)
7         return a[centro];
8     else
9         if (cliente < arreglo[centro].Π1)
10            arriba ← centro-1;
11        else
12            abajo ← centro+1;
13        endIf
14    endIf
15 endWhile

```

**Complejidad**  $O(\log(|a|))$  porque es una implementación del algoritmo de búsqueda, que por lo visto en clase, tiene complejidad logarítmica en la longitud del arreglo.

#### 7.4. Servicios Usados

Módulo	Operación	Complejidad Requerida
diccString(infoTitulo)	CrearIt	$O(1)$
diccString(infoTitulo)	Definir	$ nt $
diccString(infoTitulo)	Obtener	$ nt $
conj(promesa)	Vacio	$O(1)$
conj(promesa)	AgregarRapido	$O(1)$
itDicc(diccString(infoTitulo))	HayMás	$O(1)$
itDicc(diccString(infoTitulo))	Actual	$O(1)$
itDicc(diccString(infoTitulo))	Avanzar	$O(1)$
	BuscarCliente	$O(\log(C))$
conjEstNat	NuevoConjEstNat	$O(C(\log(C)))$
itConjEstNat	CrearIt	$O(1)$
itConjEstNat	HayProx	$O(1)$
itConjEstNat	Proximo	$O(1)$
itConjEstNat	Actual	$O(1)$
arreglo_dimensionable	CrearNuevo	$O(n)$
arreglo_dimensionable	AgregarElemento	$O(1)$
arreglo_dimensionable	•[•]	$O(1)$
	heapsort	$O(n(\log(n)))$