



**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. Introducción</b>  | <b>3</b>  |
| 1.1. Renombres de Módulos   | 3         |
| <b>2. Módulo Wolfie</b>   | <b>3</b>  |
| 2.1. Interfaz   | 3         |
| 2.1.1. Operaciones básicas de wolfie  | 3         |
| 2.1.2. Operaciones básicas de itTítulos   | 4         |
| 2.2. Representación   | 5         |
| 2.2.1. Representación de wolfie   | 5         |
| 2.2.2. Invariante de representación de wolfie                                       | 5         |
| 2.2.3. Función de abstracción de wolfie   | 6         |
| 2.2.4. Representación de itTítulos  | 7         |
| 2.2.5. Invariante de Representación de itTítulos                                    | 7         |
| 2.2.6. Función de abstracción 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>11</b> |
| 3.1. Interfaz   | 11        |
| 3.1.1. Parámetros formales  | 11        |
| 3.1.2. Operaciones básicas de Diccionario String( $\alpha$ )                        | 11        |
| 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 dicString                                    | 12        |
| 3.2.3. Operaciones auxiliares del invariante de Representación                      | 12        |
| 3.2.4. Función de abstracción de dicString  | 13        |
| 3.2.5. Representación del iterador de Claves del Diccionario String( $\alpha$ )     | 13        |
| 3.3. Algoritmos   | 14        |
| 3.3.1. Algoritmos de Diccionario String   | 14        |
| 3.3.2. Algoritmos del iterador de claves del Diccionario String                     | 15        |
| 3.4. Servicios Usados   | 15        |
| <b>4. Módulo Conjunto Estático de Nats</b>  | <b>15</b> |
| 4.1. Interfaz   | 15        |
| 4.1.1. Operaciones básicas de conjEstNat  | 15        |
| 4.1.2. Operaciones básicas de itConjEstNat  | 16        |
| 4.2. Representación   | 16        |

|           |  |           |
|-----------|--|-----------|
| 4.2.1.    | Representación de conjEstNat                 | 16        |
| 4.2.2.    | Invariante de representación de conjEstNat   | 16        |
| 4.2.3.    | Función de abstracción de conjEstNat         | 16        |
| 4.2.4.    | Representación de itConjEstNat               | 17        |
| 4.2.5.    | Invariante de representación de itConjEstNat | 17        |
| 4.2.6.    | Función de abstracción de itConjEstNat       | 17        |
| 4.3.      | Algoritmos                                   | 17        |
| 4.3.1.    | Algoritmos de conjEstNat                     | 17        |
| 4.3.2.    | Algoritmos de itConjEstNat                   | 17        |
| 4.4.      | Servicios Usados                             | 18        |
| 4.5.      | TAD CONJUNTO ESTÁTICO DE NATS                | 18        |
| <b>5.</b> | <b>Módulo Promesa</b>                        | <b>19</b> |
| 5.1.      | Interfaz                                     | 19        |
| 5.1.1.    | Operaciones básicas de promesa               | 19        |
| 5.2.      | Representación                               | 19        |
| 5.2.1.    | Representación de promesa                    | 19        |
| 5.2.2.    | Invariante de Representación de Promesa      | 19        |
| 5.2.3.    | Función de abstracción de Promesa            | 20        |
| 5.3.      | Algoritmos                                   | 20        |
| 5.3.1.    | Algoritmos de promesa                        | 20        |
| <b>6.</b> | <b>Módulo Título</b>                         | <b>20</b> |
| 6.1.      | Interfaz                                     | 20        |
| 6.1.1.    | Operaciones básicas de título                | 20        |
| 6.2.      | Representación                               | 21        |
| 6.2.1.    | Representación de título                     | 21        |
| 6.2.2.    | Invariante de Representación de Título       | 21        |
| 6.2.3.    | Función de abstracción de Título             | 21        |
| 6.3.      | Algoritmos                                   | 22        |
| 6.3.1.    | Algoritmos de título                         | 22        |

## 1. Introducción

En los distintos módulos se asume que los tipos básicos se pasan por copia, con complejidad  $O(1)$ . El tipo `STRING` se pasa también por copia, y esta operación tiene complejidad  $O(n)$ , con  $n$  la longitud del string. Para los módulos diseñados, el pasaje es por referencia. De todos modos, las aclaraciones que nos parecieron necesarias se encuentran en la sección de aliasing de la interfaz.

### 1.1. 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

**géneros**    `wolfie`, `itTítulos`

**se explica con:** `WOLFIE`, `ITERADOR UNIDIRECCIONAL`

#### 2.1.1. Operaciones básicas de `wolfie`

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

**TÍTULOS**(`in w : wolfie`)  $\rightarrow res : itUni(título)$   
**Pre**  $\equiv \{true\}$   
**Post**  $\equiv \{res =_{obs} crearItUni(títulos(w))\}$   
**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**  $\equiv \{c \in clientes(w)\}$   
**Post**  $\equiv \{res =_{obs} crearItUni(promesasDe(c, w))\}$   
**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**  $\equiv \{c \in clientes(w) \wedge (\exists t:título) (t \in títulos(w) \wedge nombre(t) = nt)\}$   
**Post**  $\equiv \{res =_{obs} accionesPorCliente(c, nt, w)\}$   
**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**  $\equiv \{res =_{obs} inaugurarWolfie(cs)\}$   
**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**  $\equiv \{w_0 =_{obs} w \wedge (\forall t2:título) (t2 \in títulos(w) \Rightarrow nombre(t) \neq nombre(t2))\}$

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

**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.

### 2.1.2. Operaciones básicas de itTítulos

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

**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**  $i$ : itTítulos)  $\rightarrow res$ : 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**  $i$ : itTítulos)  $\rightarrow res$ : [

**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**  $i$ : itTítulos)  $\rightarrow res$ : 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  $\text{tupla}(\text{títulos: diccString}(\text{infoTítulo}),$   
 $\text{clientes: conjEstNat}(\text{cliente})$   
 $\text{últimoLlamado: } \langle \text{cliente: cliente, promesas: conj}(\text{promesa}), \text{ fueÚltimo: bool} \rangle)$

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 de wolfie

- (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*  $\rightarrow$  bool

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

### 2.2.3. Función de abstracción de wolfie

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

```

damePromesas(it, e, c)  $\equiv$  if hayMas?(it) then
    if buscarCliente(Significado(actual(it), e.titulos)).promCompra  $\neq$  NULL then
        {buscarCliente(Significado(actual(it), e.titulos)).promCompra  $\neq$  NULL}  $\cup$  fi
    if buscarCliente(Significado(actual(it), e.titulos)).promVenta  $\neq$  NULL then
        {buscarCliente(Significado(actual(it), e.titulos)).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)  $\wedge_L$  esIterador(it1, Avanzar(it2)))

## 2.2.6. Función de abstracción de itTítulos

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

Abs(*i*) =<sub>obs</sub> *t*: itTítulos | actual(*t*)=Significado(actual(i.it), \*(i.dicc))  $\wedge$  (hayMás(i.it) $\Rightarrow_L$ (hayMás(*t*)  $\wedge_L$  Abs(<Avanzar(i.it),  
i.dicc>, 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)

iTítulos(in e: estr)  $\rightarrow$  res: itTítulos

1 **return** ( CrearIt ( e ) )

**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(Significado(Nombre(Actual(it)), e.titulos).arrayClientes) O(C*|nombre(actual(it))|)
7     O(1)
8     if tup.promVenta  $\neq$  NULL then AgregarRapido(proms, *(tup.promVenta)) O(1)

```



```

9      if tup.promCompra ≠ NULL then AgregarRapido(proms, *(tup.promCompra)) O(1)
10     Avanzar(it) O(1)
11   endwhile
12   e.ultimoLlamado.promesas ← proms O(1)
13 fi
14 return(crearIt(e.ultimoLlamado.promesas)) O(1)

```

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

iAccionesPorCliente(in c: cliente, in nt, nombreT, in e: estr) → res: nat

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

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

iInaugurarWolfie(in c: conj(cliente)) → res: estr

```

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

```

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

iAgregarTítulo(in t: título, in/out e: estr) → res: nat

```

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

```

**Complejidad:**  $O(|\text{nombre}(t)|+C)$

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

```

1 infoTitulo s ← Significado(nt, e.titulos) O(|nt|)
2 recotizar(cot, s.titulo)
3 nat i ← 0 O(1)
4 while i < |s.arrayClientes| C*
5   if (s.arrayClientes[i].promVenta ≠ NULL \yluego limite(*(s.arrayClientes[i].promVenta))
6> cotizacion(s.titulo)) then
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 ← 0 O(1)
16 while i < |s.arrayClientes| C*
17   if (arr[i].promCompra ≠ NULL \yluego limite(*(arr[i].promCompra)) < cotizacion(s.titulo))
18 cantidad(*(arr[i].promCompra)) ≤ 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:**  $O(|\text{nt}|)+2*O(1)+C*4*O(1)+O(C)+O(C)+O(C(\log(C)))+O(1)+C*4*O(1)+O(C)+O(C(\log(C)))=$   
 $O(|\text{nt}|+C(\log(C)))$

**Aclaraciones:** En este algoritmo hicimos un cambio de tipo de tupla a tuplaPorCantAcc, que es igual pero con otra relación de orden, por lo que al ordenar el arreglo, queda ordenado de mayor a menor por la cantidad de acciones de cada cliente. Luego de efectuar el cumplimiento de las promesas de compra, volvemos a cambiar a tuplaPorClientes y reordenamos en base a los clientes para poder seguir haciendo búsqueda logarítmica por clientes.

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

```

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

```

**Complejidad:**  $O(1)+O(1)+O(|\text{titulo}(p)|+C)=O(|\text{titulo}(p)|+C)$

iEnAlza(in nt: nombreT, in e: estr) → res: bool

```

1  return(enAlza(Significado(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                                         O(1)
3  do                                                n*
4    arr[i] ← <Actual(it), 0, NULL, NULL>           O(1)
5    i++                                             O(1)
6    Proximo(it)                                    O(1)
7  while hayProx(it)                                O(1)
8  return arr

```

**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                                         O(1)
2  while i < |a1|                                   |a1|*
3    a2[i].cliente ← a1[i].cliente                 O(1)
4    a2[i].cantAcc ← a1[i].cantAcc                 O(1)
5    a2[i].promCompra ← a1[i].promCompra           O(1)
6    a2[i].promVenta ← a1[i].promVenta             O(1)
7    i++                                             O(1)
8  endWhile

```

**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  endWhile

```

$O(1)$   
 $|a1| * O(1)$   
 $O(1)$   
 $O(1)$   
 $O(1)$   
 $O(1)$

**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 implementacion 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) | Significado     | $ nt $                |
| conj(promesa)          | Vacio           | $O(1)$                |
| conj(promesa)          | AgregarRapido   | $O(1)$                |
| itClaves(diccString)   | HayMás          | $O(1)$                |
| itClaves(diccString)   | Actual          | $O(1)$                |
| itClaves(diccString)   | 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( $\alpha$ )

El módulo Diccionario String provee un diccionario en el que la característica distintiva es que las claves definidas son strings, permitiendo esto que las operaciones de definición de claves en complejidad  $O(|clave|)$ , como así también la obtención del valor asociado a la clave y el testeo de si dicha clave está definida. Para el recorrido sus elementos se provee de un iterador de Claves que permite recorrer las claves del Diccionario String.

#### 3.1. Interfaz

##### 3.1.1. Parámetros formales

**género**  $\alpha$

**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  $d : \text{diccString}(\alpha)$ , 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 de 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:**  $O(\text{longitud}(\text{secuSuby}(d)))$

**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))

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

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

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

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

## 3.2. Representación

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

**diccString( $\alpha$ ) se representa con estrDic**

donde **estrDic** es **tupla**(*raíz*: puntero(nodo) *claves*: lista(string))

**Nodo se representa con estrNodo**

donde **estrNodo** es **tupla**(*valor*: puntero( $\alpha$ ) *hijos*: arreglo\_estático[256](puntero(nodo)) )

### 3.2.2. Invariante de Representación de 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.

**Rep** : estrDic  $\rightarrow$  bool

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

$\text{raíz} \neq \text{NULL} \wedge_{\text{L}} \text{noHayCiclos}(e) \wedge \text{todasLasHojasTienenValor}(e) \wedge$

$\text{hayHojas}(e) \Rightarrow |e.\text{claves}| > 0 \wedge$

$(\forall c : \text{string})(c \in \text{caminosANodos}(e)) \Rightarrow ((\exists i : \text{nat})(i \in \{0..|e.\text{claves}|\}) \Rightarrow (e.\text{claves}[i] = c))$

### 3.2.3. Operaciones auxiliares del invariante de Representación

**noHayCiclos** : puntero(nodo)  $\rightarrow$  bool

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

```

leer : puntero(nodo) × string → bool
leer(p, s) ≡ if vacia?(s) then
    p → valor
else
    if p → hijos[prim(s)] = NULL then NULL else leer(p → hijos[prim(s)], fin(s)) fi
fi

todosNull : arreglo(puntero(nodo)) → bool
todosNull(a) ≡ auxTodosNull(a, 0)
auxTodosNull : arreglo(puntero(nodo)) × nat → bool
auxTodosNull(a, i) ≡ if i < |a| then a[i] == NULL ∧ auxTodosNull(a, i + 1) else a[i].valor == NULL 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

hayHojas : puntero(nodo) → bool

```

Dada una estructura, indica si en la misma existe algún nodo cuyo arreglo hijos tenga todas las posiciones NULL.

```

caminosANodos : puntero(nodo) → conj(string)

```

Dada una estructura, devuelve un conjunto con el único camino existente entre la raíz y cada hoja. El camino se obtiene de agregar la posición del arreglo hijos por la cual hay que bajar en cada nivel de la estructura hasta llegar a la hoja, convirtiendo en cada paso esa posición en un CHAR y juntándolos en un STRING.

### 3.2.4. Función de abstracción de diccString

```

Abs : estrDicc e → dicc(string, α) {Rep(e)}
Abs(e) =obs d: dicc(string, α) | (∀ c:string)(definido?(c, d)) = (∃ n:nodo)(n ∈ todasLasHojas(e)) n.valor != NULL
    ∧ (∃ i:nat)(i ∈ {0..|e.claves|}) ⇒ e.claves[i] = c ∧L significado(c, d) = leer(e.clave).valor

```

### 3.2.5. Representación del iterador de Claves del Diccionario String(α)

itClaves(string) se representa con puntero(nodo)

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

### 3.3. Algoritmos

#### 3.3.1. Algoritmos de Diccionario String

ICREARDICC() → res = estrDicc( $\alpha$ )

|                   |      |
|-------------------|------|
| 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( $\alpha$ ): 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])]              | O(1)   |
| 9 i++                                 | O(1)   |
| 10 endwhile                           |        |
| 11 p.valor ← a                        | O(1)   |
| 12 agregarAdelante(hijos, c)          | O( s ) |

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

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

|                              |      |
|------------------------------|------|
| 1 i ← 0                      | O(1) |
| 2 p ← d.raiz                 | O(1) |
| 3 while (i < (longitud(s)))  | s *  |
| 4     p ← p.hijos[ord(s[i])] | O(1) |
| 5 i++                        | O(1) |
| 6 endwhile                   |      |
| 7 return p.valor             | O(1) |

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

IDEFINIDO?(in estrDicc( $\alpha$ ): d, in string: c) → res = bool

|                                       |      |
|---------------------------------------|------|
| 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         p ← p.hijos[ord(s[i])]      | O(1) |
| 6         i++ O(1)                    |      |
| 7     else                            |      |
| 8         return false                | O(1) |
| 9     endif                           |      |
| 10 endwhile                           |      |
| 11 return p.valor != NULL             | O(1) |

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

```
iCLAVES(in estrDicc( $\alpha$ ): d)  $\rightarrow$  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 | $O(\text{copy}(\alpha))$ |
| lista            | •[•]            | $O(1)$                   |

## 4. Módulo Conjunto Estático de Nats

El módulo conjunto estático de nats representa a un conjunto de naturales, con la diferencia de que no se le pueden agregar y quitar elementos, sino que se inicializa con una cantidad fija de naturales. Además, el iterador recorre el conjunto de menor a mayor.

### 4.1. Interfaz

**géneros** conjEstNat, itConjEstNat

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

#### 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)

#### 4.2.2. Invariante de representación de conjEstNat

Rep: los elementos estan ordenados y no hay repeticiones

Rep : array  $\rightarrow$  bool

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

#### 4.2.3. Función de abstracción de conjEstNat

Abs : array  $a \rightarrow$  conjEstNat

{Rep( $a$ )}

Abs( $a$ )  $=_{\text{obs}} c$ : 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**

#### 4.2.4. Representación de itConjEstNat

itConjEstNat se representa con iterador

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

#### 4.2.5. Invariante de representación de itConjEstNat

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

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

#### 4.2.6. Función de abstracción de itConjEstNat

$\text{Abs} : \text{iterador } it \rightarrow \text{itConjEstNat}$

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

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

### 4.3. Algoritmos

#### 4.3.1. Algoritmos de conjEstNat

```

iNuevoConjEstNat(in c: conj(nat)) → res: array
1  itConj(nat) it ← crearIt(c)                                O(1)
2  arreglo_dimensionable(nat)[cardinal(c)] a                 O(n)
3  nat i ← 0                                                  O(1)
4  while (HaySiguiente?(it))                                  n*
5      a[i] ← Siguiente(it)                                    O(1)
6      i++                                                    O(1)
7      Avanzar(it)                                             O(1)
8  endWhile
9  heapsort(a)                                                O(n(log(n)))
10 return(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  bool b ← false                                            O(1)
2  nat i ← 0                                                  O(1)
3  while (i < |c|)                                           n*
4      b ← (b ∨ c[i] = n)                                    O(1)
5      i++                                                    O(1)
6  endWhile
7  return(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:**  $O(1)$

#### 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** conjEstNat

**exporta** conjEstNat, generadores, observadores, #

**usa** BOOL, NAT, CONJUNTO(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

**géneros**    promesa

**se explica con:** PROMESA.

#### 5.1.1. Operaciones básicas de promesa

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

**TIPO**(**in**  $p$ : promesa)  $\rightarrow res$  : 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

**LIMITE**(**in**  $p$ : promesa)  $\rightarrow res$  : 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)  $\rightarrow res$  : 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)  $\rightarrow res$  : estr  
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{crearPromesa}(t, tipo, n, m)\}$   
**Complejidad:**  $(|t|)$   
**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 *límite*: dinero *cantidad*: nat )

#### 5.2.2. Invariante de Representación de Promesa

$\text{Rep} : \text{estr} \rightarrow \text{bool}$   
 $\text{Rep}(e) \equiv \text{true} \iff \text{true}$

### 5.2.3. Función de abstracción de Promesa

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

## 5.3. Algoritmos

### 5.3.1. Algoritmos de promesa

$\text{iTítulo}(\text{in } p : \text{estr}) \rightarrow \text{res} = \text{nombre}$   
 1  $\text{res} \leftarrow e.\text{título}$   $O(|p.\text{título}|)$

**Complejidad:**  $O(|p.\text{título}|)$

$\text{iTipo}(\text{in } p : \text{estr}) \rightarrow \text{res} = \text{tipoPromesa}$   
 1  $\text{res} \leftarrow e.\text{tipo}$   $O(1)$

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

$\text{iLímite}(\text{in } p : \text{estr}) \rightarrow \text{res} = \text{dinero}$   
 1  $\text{res} \leftarrow e.\text{límite}$   $O(1)$

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

$\text{iCantidad}(\text{in } p : \text{estr}) \rightarrow \text{res} = \text{nat}$   
 1  $\text{res} \leftarrow e.\text{cantidad}$   $O(1)$

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

$\text{iCrearPromesa}(\text{in } t : \text{nombreT}, \text{in } \text{tipo} : \text{TipoPromesa}, \text{in } n : \text{dinero}, \text{in } c : \text{nat}) \rightarrow \text{res} = \text{estr}$   
 1  $\text{res.título} \leftarrow t$   $O(|p.\text{título}|)$   
 2  $\text{res.tipo} \leftarrow \text{tipo}$   $O(1)$   
 3  $\text{res.límite} \leftarrow n$   $O(1)$   
 4  $\text{res.cantidad} \leftarrow m$   $O(1)$

**Complejidad:**  $3 * O(1) + O(|p.\text{título}|) = O(|p.\text{título}|)$

## 6. Módulo Título

### 6.1. Interfaz

**géneros**    **título**

**se explica con:** TÍTULO.

#### 6.1.1. Operaciones básicas de título

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

$\text{\#MÁXACCIONES}(\text{in } t : \text{título}) \rightarrow \text{res} : \text{nat}$   
**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \#m\acute{a}xAcciones(t)\}$

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

**Descripcion:** Devuelve el mximo de cantidad de acciones

COTIZACIN(**in**  $t$ : ttulo)  $\rightarrow res$  : dinero

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

**Post**  $\equiv \{res =_{\text{obs}} cotizacin(t)\}$

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

**Descripcion:** Devuelve la cotizacin del ttulo

ENALZA(**in**  $t$ : ttulo)  $\rightarrow res$  : bool

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

**Post**  $\equiv \{res =_{\text{obs}} enAlza(t)\}$

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

**Descripcion:** Indica si el ttulo est o no en alza

CREARTTULO(**in**  $t$ : nombre, **in**  $c$ : dinero, **in**  $n$ : nat)  $\rightarrow res$  : ttulo

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

**Post**  $\equiv \{res =_{\text{obs}} crearTtulo(s, c, n)\}$

**Complejidad:**  $O(copy(nombre(t)))$

**Descripcion:** Devuelve una nuevo ttulo

RECOTIZAR(**in**  $d$ : dinero, **in**  $t$ : ttulo)  $\rightarrow res$  : ttulo

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

**Post**  $\equiv \{res =_{\text{obs}} recotizar(d, t)\}$

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

**Descripcion:** Cambia la cotizacin del ttulo

## 6.2. Representacin

### 6.2.1. Representacin de ttulo

promesa se representa con **estr**

donde **estr** es  $tupla(nombre: nombre, \#m\acute{a}xAcciones: nat, cotizacin: dinero, enAlza: bool)$

### 6.2.2. Invariante de Representacin de Ttulo

$Rep : estr \rightarrow bool$

$Rep(e) \equiv true \iff true$

### 6.2.3. Funcin de abstraccin de Ttulo

$Abs : estr\ e \rightarrow ttulo$

$\{Rep(e)\}$

$Abs(e) =_{\text{obs}} t: ttulo \mid nombre(t) = e.nombre \wedge \#m\acute{a}xAcciones(t) = e.\#m\acute{a}xAcciones \wedge cotizacin(t) = e.cotizacin \wedge enAlza(t) = e.enAlza$

## 6.3. Algoritmos

### 6.3.1. Algoritmos de título

iNombre(in estr: t) → res = nombre

```
1 res ← e.nombre
```

$O(\text{copy}(\text{nombre}(t)))$

**Complejidad:**  $O(\text{copy}(\text{nombre}(t)))$

i#máxAcciones(in estr: t) → res = nat

```
1 res ← e.#maxAcciones
```

$O(1)$

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

iCotización(in estr: t) → res = dinero

```
1 res ← e.cotizacion
```

$O(1)$

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

iEnAlza(in estr: t) → res = bool

```
1 res ← e.enAlza
```

$O(1)$

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

iCrearTítulo(in nombre: n, in nat: max, in dinero: c) → res = estr

```
1 res.nombre ← n
```

```
2 res.#maxAcciones ← max
```

```
3 res.enAlza ← true
```

```
4 res.cotizacion ← c
```

$O(\text{copy}(\text{nombre}(t)))$

$O(1)$

$O(1)$

$O(1)$

**Complejidad:**  $O(\text{copy}(\text{nombre}(t))) + 3 \cdot O(1) = O(\text{copy}(\text{nombre}(t)))$

iRecotizar(in dinero: c, in/out estr: t)

```
1 t.enAlza ← (c > t.cotizacion)
```

```
2 t.cotizacion ← c
```

$O(1)$

$O(1)$

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