

# Algoritmos y Estructura de Datos II

Primer cuatrimestre 2014

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

## Trabajo Practico 2

### Grupo 10

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

### Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

# Índice

<b>1. Renombres de Módulos</b>	<b>4</b>
<b>2. Módulo Wolfe</b>	<b>4</b>
2.1. Interfaz	4
2.1.1. Parámetros formales	4
2.1.2. Operaciones básicas de wolfe	4
2.2. Representación	5
2.2.1. Representación de wolfe	5
2.2.2. Invariante de representación	5
2.2.3. Función de abstracción	6
2.3. Algoritmos	7
2.3.1. Funciones auxiliares	8
2.4. Servicios Usados	9
<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( $\alpha$ )	10
3.2. Representación	11
3.2.1. Representación del Diccionario String( $\alpha$ )	11
3.2.2. Operaciones auxiliares del invariante de Representación	11
3.2.3. Función de abstracción	12
3.2.4. Representación del iterador de Claves del Diccionario String( $\alpha$ )	12
3.3. Algoritmos	12
3.3.1. Algoritmos de Diccionario String	12
3.3.2. Algoritmos del iterador de claves del Diccionario String	13
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	14
4.1.2. Operaciones básicas de itConjEstNat	14
4.2. Representación	15
4.2.1. Representación de conjEstNat	15
4.2.2. Función de abstracción	15
4.2.3. Representación de itConjEstNat	15
4.2.4. Función de abstracción	15
4.3. Algoritmos	16
4.3.1. Algoritmos de conjEstNat	16
4.3.2. Algoritmos de itConjEstNat	16

4.4. Servicios Usados . . . . .	17
4.5. TAD CONJUNTO ESTÁTICO DE NATS . . . . .	17
<b>5. Módulo Promesa</b>	<b>17</b>
5.1. Interfaz . . . . .	17
5.1.1. Parámetros formales . . . . .	17
5.1.2. Operaciones básicas de promesa . . . . .	17
5.2. Representación . . . . .	18
5.2.1. Representación de promesa . . . . .	18
5.3. Algoritmos . . . . .	18
5.3.1. Algoritmos de promesa . . . . .	18
<b>6. Módulo Título</b>	<b>19</b>
6.1. Interfaz . . . . .	19
6.1.1. Operaciones básicas de título . . . . .	19
6.2. Representación . . . . .	20
6.2.1. Representación de título . . . . .	20
6.3. Algoritmos . . . . .	20
6.3.1. Algoritmos de título . . . . .	20

## 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.

## 2.2. Representación

### 2.2.1. Representación de wolfie

wolfie se representa con estr

donde estr es tupla( $\text{títulos}$ : diccString(infoTítulo),  
 $\text{clientes}$ : conjEstNat(cliente)  
 $\text{últimoLlamado}$ : <cliente: cliente, promesas:conj(promesa), fueÚltimo: bool>))

donde infoTítulo es tupla( $\text{arrayClientes}$ : array\_dimensionable(tuplaPorCliente),  $\text{título}$ : título,  $\text{accDisponibles}$ : nat )

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

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

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  $\rightarrow$  bool

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

### 2.2.2. Función de abstracción

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

```

damePromesas(it, e, c) ≡ if hayMas?(it) then
    if buscarCliente(obtener(actual(it)).promCompra ≠ NULL then
        {buscarCliente(obtener(actual(it)).promCompra ≠ NULL} ∪ fi
    if buscarCliente(obtener(actual(it)).promVenta ≠ NULL then
        {buscarCliente(obtener(actual(it)).promVenta ≠ NULL} ∪ fi
    damePromesas(avanzar(it), e, c)
else
    vacio
fi

]

```

## 2.3. Algoritmos

iClientes(in e: estr) → res: itConjEstNat

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

**Complejidad:** O(1)

iPromesasDe(in c: cliente, in/out e: estr) → res: itConj(promesa)

```

1 if ¬(e.ultimoLlamado.cliente = c ∧ e.ultimoLlamado.fueUltimo) then O(1)
2   itClaves(diccString(infoTitulo)) it ← CrearIt(e.titulos) O(1)
3   conj(promesa) proms ← vacio() O(1)
4   tuplaPorClientes tup O(1)
5   while (HayMas?(it)) T* O(1)
6     tup ← BuscarCliente(Obtener(Nombre(Actual(it)), e.titulos).arrayClientes)
7     O(C*|nombre(actual(it))|)
8     if tup.promVenta ≠ 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:**  $4*O(1)+T*(O(1)+O(C*|nombre(actual(it))|))+3*O(1))+O(1)+O(1)\subseteq O(T*C*|max\_nt|)$

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

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

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

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

iAgregarTitulo(in t: titulo, in/out e: estr) → res: nat

```

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

```

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

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

```

1 infoTitulo s ← Obtener(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:**  $O(|nt|) + 2*O(1) + C*4*O(1) + O(C) + O(C) + O(C(\log(C))) + O(1) + C*4*O(1) + O(C) + O(C(\log(C))) = O(|nt| + C(\log(C)))$

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

```

1  promesa prom ← p                                                            O(1)
2  if tipo(prom)=compra then                                                  O(1)
3    BuscarCliente(c, Obtener(titulo(prom), e.titulos).arrayClientes).promCompra ← &prom
4                                                                O(|titulo(p)|+C)
5  else
6    BuscarCliente(c, Obtener(titulo(prom), e.titulos).arrayClientes).promCompra ← &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) → res: bool

```

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

```

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

### 2.3.1. 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                                     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:**  $O(1) + |a1| * 5 * O(1) = O(|a1|)$

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**

## 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  $d : \text{diccString}(\alpha)$ , 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:**  $O(\text{longitud}(c))$

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

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

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

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

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

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

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

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

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

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

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

## 3.2. Representación

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

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

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

Nodo se representa con  $\text{estrNodo}$

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

- (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.

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

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

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

### 3.2.2. Operaciones auxiliares del invariante de Representación

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

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

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

$\text{leer}(p, s) \equiv \text{if vacia?}(s) \text{ then}$

$p \rightarrow \text{valor}$

**else**

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

**fi**

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

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

```

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

```

### 3.2.3. Función de abstracción

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

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

$\text{itClaves}(\text{string})$  se representa con puntero(nodo)

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

## 3.3. Algoritmos

### 3.3.1. Algoritmos de Diccionario String

```

iCREARDICC() → res = diccString(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
7  valor ← NULL

```

O(1)  
O(1)

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

```

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

```

O(1)  
O(1)  
|s|\*  
O(1)  
O(1)  
O(1)  
O(1)  
O(1)  
O(1)  
O(1)  
O(1)  
O(|s|)

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

```

iSIGNIFICADO(in diccString( $\alpha$ ): 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)  
O(1)

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

```

iDEFINIDO?(in diccString( $\alpha$ ): 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)  
O(1)  
O(1)

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

```

iCLAVES(in diccString( $\alpha$ ): 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	$O(\text{copy}(\alpha))$
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**  $\equiv \{hayMas?(i)\}$   
**Post**  $\equiv \{res =_{obs} avanzar(i)\}$   
**Complejidad:**  $O(1)$   
**Descripcion:** Avanza el iterador

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

### 4.2.2. Función de abstracción

Abs : array  $a \rightarrow$  conjEstNat

{Rep( $a$ )}

Abs( $a$ )  $=_{obs} c$ : conjEstNat |  $(\forall n: nat) n \in c \iff estáEnArray?(n, a, 0)$

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

estáEnArray( $n, a, i$ )  $\equiv$  **if**  $i = longitud(a)-1$  **then** false **else**  $a[i] = n \vee estáEnArray?(n, a, i+1)$  **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( $i$ )  $\equiv true \iff i.pos < longitud(*i.lista)$

### 4.2.4. Función de abstracción

Abs : iterador  $it \rightarrow$  itConjEstNat

{Rep( $it$ )}

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

### 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))
5      a[i] ← Siguiente(it)                                    n*
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|)
4      b ← (b ∨ c[i]=n)                                       n*
5      i++                                                    O(1)
6  endWhile                                                  O(1)
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 (<|c|, &c>)

```

**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 básicos, incluidos arreglos y punteros.



#### 4.4. Servicios Usados

Módulo	Operación	Complejidad Requerida
arreglo_estático	CrearNuevo	$O(n)$
arreglo_estático	AgregarElemento	$O(1)$
arreglo_estático	$\bullet[\bullet]$	$O(1)$
	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

#### 5.1.1. Parámetros formales

**géneros** promesa

**se explica con:** PROMESA.

#### 5.1.2. Operaciones básicas de promesa

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

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

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

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

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

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

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

**CANTIDAD**(*in p: promesa*)  $\rightarrow res : cantidad$   
**Pre**  $\equiv \{true\}$   
**Post**  $\equiv \{res =_{obs} cantidad(p)\}$   
**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**  $\equiv \{true\}$   
**Post**  $\equiv \{res =_{obs} crearPromesa(t, tipo, n, m)\}$   
**Complejidad:** (1)  
**Descripcion:** Devuelve una nueva promesa

## 5.2. Representación

### 5.2.1. Representación de promesa

promesa se representa con *estr*

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

*Rep* : *estr*  $\rightarrow$  bool

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

*Abs* : *estr e*  $\rightarrow$  promesa

{*Rep*(*e*)}

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

## 5.3. Algoritmos

### 5.3.1. Algoritmos de promesa

*iTitulo*(*in p: estr*)  $\rightarrow res = nombre$

1 *res* = *e.titulo*

*iTipo*(*in p: estr*)  $\rightarrow res = tipoPromesa$

1 *res* = *e.tipo*

*iLimite*(*in p: estr*)  $\rightarrow res = dinero$

1 *res* = *e.limite*

*iCantidad*(*in p: estr*)  $\rightarrow res = nat$

1 *res* = *e.cantidad*

```
iCrearPromesa(in t: nombreT, in tipo: TipoPromesa, in n: dinero, in c: nat) → res = estr
```

```
1 res.titulo = t
2 res.tipo = tipo
3 res.limite = n
4 res.cantidad = m
```

## 6. Módulo Título

### 6.1. Interfaz

**géneros**    **título**

**se explica con:** TÍTULO.

#### 6.1.1. Operaciones básicas de título

**NOMBRE**(in *t*: título) → *res* : nombre  
**Pre** ≡ {true}  
**Post** ≡ {*res* =<sub>obs</sub> nombre(*t*)}  
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve el nombre del título

**#MÁXACCIONES**(in *t*: título) → *res* : nat  
**Pre** ≡ {true}  
**Post** ≡ {*res* =<sub>obs</sub> #máxAcciones(*t*)}  
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve el máximo de cantidad de acciones

**COTIZACIÓN**(in *t*: título) → *res* : dinero  
**Pre** ≡ {true}  
**Post** ≡ {*res* =<sub>obs</sub> cotización(*t*)}  
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve la cotización del título

**ENALZA**(in *t*: título) → *res* : bool  
**Pre** ≡ {true}  
**Post** ≡ {*res* =<sub>obs</sub> enAlza(*t*)}  
**Complejidad:**  $O(1)$   
**Descripcion:** Indica si el título está o no en alza

**CREARTÍTULO**(in *t*: nombre, in *c*: dinero, in *n*: nat) → *res* : título  
**Pre** ≡ {true}  
**Post** ≡ {*res* =<sub>obs</sub> crearTítulo(*s*, *c*, *n*)}  
**Complejidad:**  $O(1)$   
**Descripcion:** Devuelve una nuevo título

**RECOTIZAR**(in *d*: dinero, in *t*: título) → *res* : título  
**Pre** ≡ {true}

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

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

**Descripcion:** Cambia la cotización del título

## 6.2. Representación

### 6.2.1. Representación de título

promesa se representa con **estr**

donde **estr** es `tupla(nombre: nombre, #máxAcciones: nat, cotización: dinero, enAlza: bool)`

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

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

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

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

$\text{Abs}(e) =_{\text{obs}} t: \text{título} \mid \text{nombre}(t) = e.\text{nombre} \wedge \#máxAcciones(t) = e.\#máxAcciones \wedge \text{cotizacion}(t) = e.\text{cotizacion} \wedge \text{enAlza}(t) = e.\text{enAlza}$

## 6.3. Algoritmos

### 6.3.1. Algoritmos de título

$i\text{Nombre}(\text{in } \text{estr}: t) \rightarrow \text{res} = \text{nombre}$

```
1 res = e.nombre
```

$i\#máxAcciones(\text{in } \text{estr}: t) \rightarrow \text{res} = \text{nat}$

```
1 res = e.#maxAcciones
```

$i\text{Cotización}(\text{in } \text{estr}: t) \rightarrow \text{res} = \text{dinero}$

```
1 res = e.cotizacion
```

$i\text{EnAlza}(\text{in } \text{estr}: t) \rightarrow \text{res} = \text{bool}$

```
1 res = e.enAlza
```

$i\text{CrearTítulo}(\text{in } \text{nombre}: n, \text{in } \text{nat}: \text{max}, \text{in } \text{dinero}: c) \rightarrow \text{res} = \text{estr}$

```
1 res.nombre = n
```

```
2 res.#maxAcciones = max
```

```
3 res.enAlza=true
```

```
4 res.cotizacion = c
```

$i\text{Recotizar}(\text{in } \text{dinero}: c, \text{in/out } \text{estr}: t)$

```
1 if c>t.cotizacion then
```

```
2   t.enAlza=true
```

```
3 else
```

```
4   t.enAlza=false
```

```
5 fi
```

```
6 t.cotizacion <- c
```