Algoritmos y Estructuras de Datos II

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

Trabajo Práctico 2: Diseño

Wolfie

Grupo 16

Integrante	LU	Correo electrónico
Agustina Aldasoro	86/13	agusaldasoro@hotmail.com
Nicolás Chamo	282/13	nicochamo@hotmail.com
Belén Bouzón	128/13	belenbouzon@hotmail.com
Alexander Ledezma	337/12	lralexandr@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Módulo Título

1.1. Servicios Exportados

Operacion	Aliasing	Complejidad
Nombre	Si	O(1)
#maxAcciones	No	O(1)
Cotización	No	O(1)
EnAlza	No	O(1)
CrearTítulo	No	O(1)
Recotizar	No	O(1)
CopiarTítulo	No	O(|nombre(t)|)

1.2. Interfaz

Interfaz TÍTULO
Se explica con: Título
Géneros: título
Operaciones:

$Nombre(\mathbf{in}\, t: Titulo) \rightarrow resultado$: Nombre
Pre $\equiv \{\ true\ \}$
Post $\equiv \{\ \widehat{resultado} =_{\text{obs}} Nombre(\hat{t})\ \}$

$\#maxAcciones(\mathbf{in}\, t: Titulo) \rightarrow resultado$: Nat
Pre $\equiv \{\ true\ \}$
Post $\equiv \{\ \widehat{resultado} =_{\text{obs}} \#maxAcciones(\hat{t})\ \}$

$Cotizacion(\mathbf{in}\, t: Titulo) \rightarrow resultado$: Dinero
Pre $\equiv \{\ true\ \}$
Post $\equiv \{\ \widehat{resultado} =_{\text{obs}} Cotización(\hat{t})\ \}$

$enAlza(\mathbf{in}\, t: Titulo) \rightarrow resultado$: Bool
Pre $\equiv \{\ true\ \}$
Post $\equiv \{\ \widehat{resultado} =_{\text{obs}} enAlza(\hat{t})\ \}$

$CrearTítulo(\mathbf{in}\, nT: Nombre, \mathbf{in}\, d: Dinero, \mathbf{in}\, n: Nat) \rightarrow resultado$: Título
Pre $\equiv \{\ true\ \}$
Post $\equiv \{\ \widehat{resultado} =_{\text{obs}} CrearTítulo(\hat{nT}, \hat{d}, \hat{n})\ \}$

Recotizar(**in** d: *Dinero*, **inout** t: *Titulo*)

Pre $\equiv \{ \ \hat{t} =_{\text{obs}} t_0 \ \}$

Post $\equiv \left\{ \begin{array}{l} nombre(\hat{t}) =_{\text{obs}} nombre(\hat{t_0}) \wedge cotización(\hat{t}) =_{\text{obs}} d \wedge \\ \#maxAcciones(\hat{t_0}) =_{\text{obs}} \#maxAcciones(\hat{t}) \wedge enAlza(\hat{t}) =_{\text{obs}} cotización(t_0) < d \end{array} \right\}$

CopiarTitulo(**in** t: *Titulo*) $\rightarrow$ *resultado*: Título

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

Post $\equiv \left\{ \ \widehat{resultado} =_{\text{obs}} \hat{t} \ \right\}$

1.3. Estructura de Representación

Título se representa con estr
donde estr es tupla <Nombre: *String*, Cotización: *Nat*, enAlza: *Bool*, #maxAcciones: *Nat* >

1.4. Invariante de Representación

Rep: $\widehat{estr} \rightarrow$ bool
($\forall$ e: $\widehat{estr}$) Rep(e) $\Leftrightarrow$(*true*)

1.5. Función de Abstracción

Abs: $\widehat{estr}$ e $\rightarrow$ Titulo (Rep(e))
($\forall$ e: $\widehat{estr}$) Abs(e) $\equiv$ t: $\widehat{Titulo}$ /

$$\left(\begin{array}{l} nombre(t) = e.Nombre \wedge \\ cotización(t) = e.Cotización \wedge \\ \#maxAcciones(t) = e.\#maxAcciones \wedge \\ enAlza(t) = e.enAlza \end{array} \right)$$

1.6. Algoritmos

*i*NOMBRE(**in** t: *estr*)$\rightarrow$ *resultado*: *nombre*
1 **return** $t.Nombre$
2 **end function**

i#MAXACCIONES(**in** t: *estr*)$\rightarrow$ *resultado*: *nat*
1 **return** $t.\#maxAcciones$
2 **end function**

*i*COTIZACION(**in** t: *estr*)$\rightarrow$ *resultado*: *dinero*

```
1  return t.Cotizacion
2  end function
```

```
iEnAlza(in t: estr)→ resultado: bool
1  return t.EnAlza
2  end function
```

```
iCrearTitulo(in nT: Nombre, in d: Dinero, in n: Nat)→ resultado: titulo
1  resultado.Nombre ← nT
2  resultado.#maxAcciones ← n
3  resultado.Cotizacion ← d
4  resultado.EnAlza ← true
5  return resultado
6  end function
```

```
iRecotizar(in d: Dinero, inout t: estr)
1  if (d > t.Cotizacion)
2     then
3           t.EnAlza ← true;
4     else
5           t.EnAlza ← false;
6  end if
7  t.Cotizacion ← d
8
9  end function
```

```
iCopiarTitulo(in t: estr)→ resultado: estr
1  resultado.Nombre ← t.Nombre
2  resultado.#maxAcciones ← t.#maxAcciones
3  resultado.Cotizacion ← t.Cotizacion
4  resultado.EnAlza ← t.EnAlza
5  return resultado
6  end function
```

2. Módulo Informacion de Titulo

2.1. Especificación

TAD INFORMACION DE TITULO

igualdad observacional

$$(\forall infT1, infT2 : \text{infoTitulo}) \left(infT1 =_{\text{obs}} infT2 \iff \begin{array}{c} (titulo(infT1) =_{\text{obs}} titulo(infT2) \wedge \\ ventas(infT1) =_{\text{obs}} ventas(infT2) \wedge \\ compras(infT1) =_{\text{obs}} compras(infT2) \wedge \\ disponibles(infT1) =_{\text{obs}} disponibles(infT2) \end{array} \right)$$

géneros infoTitulo

exporta generadores y observadores

usa PROMESA, CLIENTE, NAT, CONJ(INFOPROM), ARREGLO_ORDENABLE(INFOPROM)

observadores básicos

titulo : infoTitulo $\longrightarrow$ titulo

Ventas : infoTitulo $\longrightarrow$ conj(infoProm)

Compras : infoTitulo $\longrightarrow$ arreglo_ordenable(infoProm)

disponibles : infoTitulo $\longrightarrow$ nat

generadores

nuevoinfoTitulo : titulo t $\times$ conj(infoProm) v $\times$ arreglo_ordenable(infoProm) c $\times$ nat n $\longrightarrow$ infoTit

axiomas $\forall t$: titulo, $\forall v, c$: infoProm, $\forall d$: nat

titulo(nuevoInfoTitulo(t, v, c, d)) $\equiv$ t

ventas(nuevoInfoTitulo(t, v, c, d)) $\equiv$ v

compras(nuevoInfoTitulo(t, v, c, d)) $\equiv$ c

disponibles(nuevoInfoTitulo(t, v, c, d)) $\equiv$ d

Fin TAD

2.2. Servicios Exportados

Operacion	Aliasing	Complejidad
nuevoInfoTitulo	No	O(copy(V)+copy(C))
titulo	Si	O(1)
ventas	Si	O(1)
compras	Si	O(1)
disponibles	Si	O(1)

Donde V es un conjunto de infoProm y C un arreglo ordenable de infoProm

2.3. Interfaz

Interfaz INFORMACION DE TITULO
Se explica con: TAD Informacion de Titulo
Géneros: infoTit
Operaciones:

$nuevoInfoTitulo(\mathbf{in}\, t\colon titulo, \mathbf{in}\, v\colon infoProm, \mathbf{in}\, c\colon infoProm, \mathbf{in}\, d\colon nat) \rightarrow resultado$: infoTit

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

Post $\equiv\{\ resultado =_{\text{obs}} nuevoInfoTitulo(t, v, c, d)\ \}$

$titulo(\mathbf{in}\, iT\colon infoTit) \rightarrow resultado$: titulo

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

Post $\equiv\{\ resultado =_{\text{obs}} titulo(iT)\ \}$

$ventas(\mathbf{in}\, iT\colon infoTit) \rightarrow resultado$: conj(infoProm)

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

Post $\equiv\{\ resultado =_{\text{obs}} ventas(iT)\ \}$

$compras(\mathbf{in}\, iT\colon infoTit) \rightarrow resultado$: conj(infoProm)

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

Post $\equiv\{\ resultado =_{\text{obs}} compras(iT)\ \}$

$disponibles(\mathbf{in}\, iT\colon infoTit) \rightarrow resultado$: nat

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

Post $\equiv\{\ resultado =_{\text{obs}} disponibles(iT)\ \}$

2.4. Estructura de Representación

estr es tupla<
elIterador: itConj(titulo),
Ventas: conj(infoProm),
Compras: arreglo_ordenable(infoProm)
Disponibles: nat
>

2.5. Invariante de Representación

1)Todas las promesas son del titulo al cual apunta el iterador
2)No existe un cliente que tenga dos promesas en ventas
3)No existe un cliente que tenga dos promesas en compras
4)Ventas solo tiene promesas de ventas
5)Compras solo tiene promesas de compras

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

$$(\forall \text{ e: } \widehat{estr})\ \text{Rep(e)} \Leftrightarrow \left(\begin{array}{l} ((\forall pr : infoProm)pr \in damePromesas(e.compras) \Rightarrow_{\text{L}} \\ ((tipo(promesa(pr)) == compra) \wedge (titulo(promesa(pr)) == nombre(siguiente(e.elIterador))))) \wedge \\ ((\forall pr : infoProm)pr \in e.ventas \Rightarrow_{\text{L}} \\ ((tipo(promesa(pr)) == venta) \wedge (titulo(promesa(pr)) == nombre(siguiente(e.elIterador))))) \wedge \\ ((\forall pr1, pr2 : infoProm)pr1, pr2 \in damePromesas(e.compras) \Rightarrow \\ (cliente(pr1) == cliente(pr2) \Leftrightarrow pr1 =_{\text{obs}} pr2)) \\ ((\forall pr1, pr2 : infoProm)pr1, pr2 \in e.ventas \Rightarrow \\ (cliente(pr1) == cliente(pr2) \Leftrightarrow pr1 =_{\text{obs}} pr2)) \end{array} \right)$$

2.6. Función de Abstracción

Abs: $\widehat{infoTit}$ e $\rightarrow$ $\widehat{infoTitulo}$ (Rep(e))

$$(\forall \text{ iT: } \widehat{infoTit})\ \ \text{Abs(iT)} \equiv \text{it} : \widehat{infoTit} \Leftrightarrow \left(\begin{array}{l} titulo(iT) =_{\text{obs}} (siguiente(e.elIterador)) \wedge \\ Ventas(iT) =_{\text{obs}} e.Ventas \wedge \\ Compras(iT) =_{\text{obs}} e.Compras \wedge \\ Disponibles(iT) =_{\text{obs}} e.Disponibles \end{array} \right)$$

2.7. Algoritmos

*i*NUEVOINFOTITULO(**in** pT: $itConj(titulo)$, **in** v: $conj(infoProm)$, **in** c: $arreglo\_ordenable(infoProm)$, **in** d: nat)$\rightarrow$ $resulta$

```
1  var infoT: infoTitulo
2  infoT.titulo ← pT
3  infoT.ventas ← copiar(v)
4  infoT.compras ← copiar(c)
5  infoT.disponibles ← d
6  return infoT
7  end function
```

*i*TITULO(**in** e: $estr$)$\rightarrow$ $resultado$: $titulo$

```
1  return siguiente(e.elIterador)
2  end function
```

*i*VENTAS(**in** e: $estr$)$\rightarrow$ $resultado$: $conj(infoProm)$

```
1  return e.ventas
2  end function
```

iCOMPRAS(**in** e: $estr$)→ $resultado$: $arreglo\_ordenable(infoProm)$
1 **return** $e.compras$
2 **end function**

iDISPONIBLES(**in** e: $estr$)→ $resultado$: nat
1 **return** $e.disponibles$
2 **end function**

2.8. Servicios usados

Arreglo_ordenable

Operacion	Aliasing	Complejidad
Crear	No	O(n)
DameIndice	Si	O(1)
Ordenar	No	O(n * log n)
Insertar	No	O(1)
Borrar	No	O(1)
Tamaño	No	O(1)
DamePromesas	No	O(1)

n es el largo del arreglo

Informacion de Promesa

Operacion	Aliasing	Complejidad
nuevoinfoProm	No	O(|nombre(t)|)
cliente	No	O(1)
promesa	No	O(1)
acccionesDelCliente	No	O(1)

Promesa

Operacion	Aliasing	Complejidad
Título	Si	O(1)
Tipo	No	O(1)
Límite	No	O(1)
Cantidad	No	O(1)
CrearPromesa	No	O(1)
CopiarPromesa	No	O(|nombre(t)|)

Titulo

Operacion	Aliasing	Complejidad
Nombre	Si	O(1)
#maxAcciones	No	O(1)
Cotización	No	O(1)
EnAlza	No	O(1)
CrearTítulo	No	O(1)
Recotizar	No	O(1)
CopiarTítulo	No	O(|nombre(t)|)

Usa el Modulo Conjunto Lineal de la Catedra

3. Módulo Promesa

3.1. Servicios Exportados

Operacion	Aliasing	Complejidad
Título	Si	O(1)
Tipo	No	O(1)
Límite	No	O(1)
Cantidad	No	O(1)
CrearPromesa	No	O(1)
CopiarPromesa	No	O(|nombre(t)|)

3.2. Interfaz

Interfaz PROMESA
Se explica con: Promesa
Géneros: Promesa
Operaciones:

CrearPromesa(**in** nT: *nombre*, **in** tP: *tipoPromesa*, **in** d: *dinero*, **in** n: *nat*) $\rightarrow$ *resultado*: Promesa

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

Post $\equiv \{ \ \hat{resultado} =_{\text{obs}} CrearPromesa(\hat{nT}, \hat{tP}, \hat{d}, \hat{n}) \ \}$

título(**in** p: *Promesa*) $\rightarrow$ *resultado*: Nombre

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

Post $\equiv \{ \ \hat{resultado} =_{\text{obs}} título(\hat{p}) \ \}$

tipo(**in** p: *Promesa*) $\rightarrow$ *resultado*: tipoPromesa

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

Post $\equiv \{ \ \hat{resultado} =_{\text{obs}} tipo(\hat{p}) \ \}$

límite(**in** p: *Promesa*) $\rightarrow$ *resultado*: dinero

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

Post $\equiv \{ \ \hat{resultado} =_{\text{obs}} límite(\hat{p}) \ \}$

cantidad(**in** p: *Promesa*) $\rightarrow$ *resultado*: nat

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

Post $\equiv \{ \ \hat{resultado} =_{\text{obs}} cantidad(\hat{p}) \ \}$

CopiarPromesa(**in** p: *Promesa*) $\rightarrow$ *resultado*: Promesa

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

Post $\equiv \{\ \hat{resultado} =_{\text{obs}} \widehat{p}\ \}$

3.3. Estructura de Representación

promesa se representa con estr
donde estr es tupla <título: $string$, tipo: $enum[Compra, Venta]$, límite: nat, cantidad: nat >

3.4. Invariante de Representaciión

Rep: $\widehat{estr} \rightarrow$ bool
($\forall$ e: $\widehat{estr}$) Rep(e) $\Leftrightarrow$($true$)

3.5. Función de Abstracción

Abs: $\widehat{estr}$ e $\rightarrow$ Promesa
($\forall$ e: $\widehat{estr}$) Abs(e) $\equiv p$: $Promesa$ /

$$\left(\begin{array}{l} e.titulo = titulo(p) \wedge \\ e.tipo = tipo(p) \wedge \\ e.limite = limite(p) \wedge \\ e.cantidad = cantidad(p) \end{array}\right)$$

3.6. Algoritmos

iCrearPromesa(**in** nT: $nombre$, **in** tP: $tipoPromesa$, **in** d: $dinero$, **in** n: nat)$\rightarrow$ res: $estr$
1 $res.titulo \leftarrow nT$
2 $res.tipo \leftarrow tP$
3 $res.limite \leftarrow d$
4 $res.cantidad \leftarrow n$
5 **return** res
6 **end function**

iTitulo(**in** p: $estr$)$\rightarrow$ res: $Nombre$
1 $res \leftarrow p.titulo$
2 **return** res
3 **end function**

iTipo(**in** p: $estr$)$\rightarrow$ res: $TipoPromesa$
1 $res \leftarrow p.tipo$
2 **return** res
3 **end function**

iLIMITE(**in** p: $estr$)$\rightarrow$ res: $Dinero$
1 $res \leftarrow p.limite$
2 **return** res
3 **end function**

iCANTIDAD(**in** p: $estr$)$\rightarrow$ res: Nat
1 $res \leftarrow p.cantidad$
2 **return** res
3 **end function**

iCOPIARPROMESA(**in** p: $estr$)$\rightarrow$ res: $estr$
1 $res.titulo \leftarrow Copiar(p.titulo)$
2 $res.tipo \leftarrow p.tipo$
3 $res.limite \leftarrow p.limite$
4 $res.cantidad \leftarrow p.cantidad$
5 **return** res
6 **end function**

3.7. Servicios usados

4. Módulo Informacion de Promesa

4.1. Especificación

TAD INFORMACION DE PROMESA

igualdad observacional

$$(\forall infP1, infP2 : \text{infoProm}) \left(infP1 =_{\text{obs}} infP2 \iff \begin{array}{r} (cliente(infP1) = cliente(infP2) \wedge \\ promesa(infP1) =_{\text{obs}} promesa(infP2) \wedge \\ accionesDelCliente(infP1) = \\ accionesDelCliente(infP2)) \end{array} \right)$$

géneros infoProm

exporta generadores y observadores

usa PROMESA, CLIENTE, NAT

observadores básicos

cliente : infoProm $\longrightarrow$ cliente

promesa : infoProm $\longrightarrow$ promesa

accionesDelCliente : infoProm $\longrightarrow$ nat

generadores

nuevoinfoProm : cliente c $\times$ promesa p $\times$ nat n $\longrightarrow$ infoProm

axiomas $\forall c$: cliente, $\forall p$: promesa, $\forall n$: nat

cliente(nuevoinfoProm(c, p, n)) $\equiv$ c

promesa(nuevoinfoProm(c, p, n)) $\equiv$ p

accionesDelCliente(nuevoinfoProm(c, p, n)) $\equiv$ n

Fin TAD

4.2. Servicios Exportados

Operacion	Aliasing	Complejidad
nuevoinfoProm	No	O(|nombre(t)|)
cliente	No	O(1)
promesa	No	O(1)
acccionesDelCliente	No	O(1)

4.3. Interfaz

Interfaz INFORMACION DE PROMESA
Se explica con: TAD Informacion de Promesa
Géneros: infoProm

Operaciones:

nuevoinfoProm(**in** c: *cliente*, **in** p: *promesa*, **in** n: *nat*) $\rightarrow$ *resultado*: infoProm
Pre $\equiv\{$ *true* $\}$
Post $\equiv\{$ $resultado =_{\text{obs}} nuevoinfoProm(c, p, n)$ $\}$

cliente(**in** iP: $infoProm$) $\rightarrow$ *resultado*: cliente
Pre $\equiv\{$ *true* $\}$
Post $\equiv\{$ $resultado = cliente(iP)$ $\}$

promesa(**in** iP: $infoProm$) $\rightarrow$ *resultado*: promesa
Pre $\equiv\{$ *true* $\}$
Post $\equiv\{$ $resultado =_{\text{obs}} promesa(iP)$ $\}$

accionesDelCliente(**in** iP: $infoProm$) $\rightarrow$ *resultado*: nat
Pre $\equiv\{$ *true* $\}$
Post $\equiv\{$ $resultado = accionesDelCliente(iP)$ $\}$

4.4. Estructura de Representación

Estr es tupla<cliente: cliente, promesa: promesa, accionesDelCliente: punteroANat>

4.5. Invariante de Representaciión

Rep: $\widehat{estr} \rightarrow$ bool
($\forall$ e: $\widehat{estr}$) Rep(e) $\Leftrightarrow$(*true*)

4.6. Función de Abstracción

Abs: $\widehat{infoProm}$ e $\rightarrow$ $\widehat{infoProm}$ (Rep(e))

($\forall$ iP: $\widehat{infoProm}$) Abs(iP) $\equiv$ iprom : $\widehat{infoProm}$ $\Leftrightarrow$
$$\left(\begin{array}{l} e.cliente = cliente(iprom) \wedge \\ e.promesa =_{obs} promesa(iprom) \wedge \\ (*e.accionesDelCliente) = accionesDelCliente(iprom) \end{array} \right)$$

4.7. Algoritmos

iNUEVOINFOPROM(**in** c: $cliente$, **in** p: $promesa$, **in** pN : $punteroANat$)→ $resultado$: $infoProm$
```
1  var infoP : infoProm
2  infoP.cliente ← c
3  infoP.promesa ← copiar(p)
4  infoP.accionesDelCliente ← pN
5  return infoT
6  end function
```

iCLIENTE(**in** e: $estr$)→ $resultado$: $cliente$
```
1  return e.cliente
2  end function
```

iPROMESA(**in** e: $estr$)→ $resultado$: $promesa$
```
1  return e.promesa
2  end function
```

iACCIONESDELCLIENTE(**in** e: $estr$)→ $resultado$: nat
```
1  return (*e.accionesDelCliente)
2  end function
```

4.8. Servicios usados

Promesa

Operacion	Aliasing	Complejidad
Título	Si	O(1)
Tipo	No	O(1)
Límite	No	O(1)
Cantidad	No	O(1)
CrearPromesa	No	O(1)
CopiarPromesa	No	O(|nombre(t)|)

5. Módulo Wolfie

5.1. Servicios Exportados

Operacion	Aliasing	Complejidad
InaugurarWolfie	No	$O(\#(cs)^2)$
AgregarTítulo	No	O(|nt| + C)
ActualizarCotización	No	O(C · |nt| + C · log(C))
AgregarPromesa	No	O(|título(p)| + log(c))
Clientes	Sí	O(1)
Títulos	Sí	O(1)
PromesasDe	Sí	O(T · C · |max_nt|) y O(1) para llamados consecutivos con el mismo c
AccionesPorCliente	No	O(log(C) + |nt|)

Cs es el conjunto inicial de clientes
Nt es el nombre de un título y |Nt| su longitud
P es la cantidad de promesas pendientes de Wolfie
C es la cantidad de clientes
|max_nt| es la longitud máxima entre todos los nombres de los títulos

5.2. Interfaz

Interfaz WOLFIE
Se explica con: wolfie
Géneros: wolfie
Operaciones:

inaugurarWolfie(in *cs*:conj(cliente)) $\rightarrow$ *resultado*: Wolfie

Pre $\equiv \{ \neg vacio(cs) \}$

Post $\equiv \{ resultado =_{\text{obs}} inaugurarWolfie(cs) \}$

agregarTítulo(inout w: wolfie, in t: Título)

Pre $\equiv \{ w =_{\text{obs}} w_0 \wedge (\forall t_2 : titulo)(t_2 \in titulos(w) \rightarrow nombre(t) \neq nombre(t_2)) \}$

Post $\equiv \{ w =_{\text{obs}} agregarTitulo(t, w_0) \}$

actualizarCotizacion(in *nt*: nombre, in *cot*: nat, inout w: wolfie)

Pre $\equiv \{ w =_{\text{obs}} w_0 \wedge (\exists t : titulo)(t \in titulos(w) \wedge nombre(t) = nT) \}$

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

agregarPromesa(in *c*: cliente, in *p*: promesa, inout w: wolfie)

$$\textbf{Pre} \equiv \left\{ \begin{array}{l} w =_{\text{obs}} w_0 \wedge_{\text{L}} ((\exists t : titulo)(t \in titulos(w) \wedge nombre(t) = titulo(p)) \wedge c \in clientes(w) \wedge_{\text{L}} \\ (\forall p_2 : promesa)(p_2 \in promesasDe(c, w) \Rightarrow (titulo(p) \neq titulo(p_2) \vee \\ tipo(p) \neq tipo(p_2))) \wedge (tipo(p) = vender \\ \Rightarrow accionesPorCliente(c, titulo(p), w) \geq cantidad(p))) \end{array} \right\}$$

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

clientes(in w:wolfie) $\rightarrow$ *resultado*: conj(cliente)

$\textbf{Pre} \equiv \{\ true\ \}$

$\textbf{Post} \equiv \{\ resultado =_{\text{obs}} clientes(w)\ \}$

titulos(in w:wolfie) $\rightarrow$ *resultado*: conj(titulo)

$\textbf{Pre} \equiv \{\ true\ \}$

$\textbf{Post} \equiv \{\ resultado =_{\text{obs}} titulos(w)\ \}$

promesasDe(in c:cliente, in w: wolfie) $\rightarrow$ *resultado*: conj(promesa)

$\textbf{Pre} \equiv \{\ c \in clientes(w)\ \}$

$\textbf{Post} \equiv \{\ resultado =_{\text{obs}} promesasDe(c, w)\ \}$

accionesPorCliente(in c:cliente, in nt: nombre, in w: wolfie) $\rightarrow$ *resultado*: Nat

$\textbf{Pre} \equiv \{\ c \in clientes(w) \wedge (\exists t : titulo)(t \in titulos(w) \wedge nombre(t) = nt)\ \}$

$\textbf{Post} \equiv \{\ resultado =_{\text{obs}} accionesPorCliente(c, nt, w)\ \}$

5.3. Estructura de Representación

Wolfie se representa con estr

Donde estr es tupla <diccClientesAcciones: *diccionarioInception*
diccPromesas: *diccConMemoria*
conjuntoClientes: *conjunto*(*cliente*)
conjuntoTitulos: *conjunto*(*titulo*)>

5.4. Invariante de Representación

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

$$(\forall\ e{:}\ \widehat{estr})\ \text{Rep(e)} \Leftrightarrow \left(\begin{array}{l} 1)(\neg(vacio?(e.conjuntoClientes) \wedge \\ 2)clientesDI(e.diccClientesAcciones) =_{\text{obs}} e.conjuntoClientes \wedge_{\text{L}} \\ 3)(\forall c : cliente)(c \in e.conjuntoClientes \Rightarrow_{\text{L}} \\ ((\forall nt : nombre)(nt \in titulosDe(c, e.diccClientesAciones) \Rightarrow_{\text{L}} \\ (\exists t : titulo)(t \in e.conjuntoTitulos \wedge nombre(t) = nt))))) \wedge \\ 4)(\forall nt : nombre)(nt \in titulosDM(e.diccPromesas) \Leftrightarrow \\ (\exists t : titulo)(t \in e.conjuntoTitulos)(nombre(t) = nt) \wedge \\ t =_{\text{obs}} titulo(obtenerDM(e.diccPromesas, nombre(t)))) \wedge \\ 5)(\forall t : titulo)(t \in e.conjuntoTitulos) \Rightarrow_{\text{L}} \\ (\#maxAcciones(t) \geq \\ cantidadTotalPorTitulo(e.conjuntoClientes, e.diccClientesAcciones, nombre(t))) \\ 6)(\forall nt : nombre)(nt \in titulosDM(e.diccPromesas) \Rightarrow_{\text{L}} \\ ((\forall infT : infoTitulo)(infT =_{\text{obs}} obtenerDM(nt, e.diccPromesas)) \Rightarrow_{\text{L}} \\ ((\forall infP : infoPromesa)(infP \in compras(infT) \vee infP \in ventas(infT)) \Rightarrow_{\text{L}} \\ (accionesTotales(e.diccClientesAcciones, cliente(infP)) == accionesDelCliente(infP)))) \\ 7)(\forall nt : nombre)(nt \in titulosDM(e.diccPromesas \Rightarrow_{\text{L}} \\ (\#maxAcciones(titulo(obtener(e.diccPromesas, nT))) - \\ cantidadTotalPorTitulo(e.conjuntoClientes, e, diccClientesAcciones, nT) = \\ disponibles(obtener(e.diccPromesas, nT))) \end{array} \right)$$

donde

cantidadTotalPorTitulo : cs: conj(cliente) × dicc: diccionarioInception × nt: nombre ⟶ nat

cantidadTotalPorTitulo(cs,dicc, nt) ≡ **if** vacio?(cs) **then** 0
else
if nt ∈ titulosDe(dameUno(cs)) **then**
cantidad(c, nt, dicc) + cantidadTotalPorTitulo(sinUno(cs),
dicc, nt)
else
cantidadTotalPorTitulo(sinUno(cs), dicc, nt)
fi
fi

Explicación:
1) El conjunto de clientes no es vacío.
2) El conjunto de clientes de Wolfie es igual observacionalmente a las claves de su diccionario de tipo Inception.
3) Todos los nt asociados a cada cliente de Wolfie son nombres de títulos válidos incluídos en el conjunto de títulos de la estructura.
4) Todos los nombres de títulos que figuran como claves en el diccionario Con Memoria de Wolfie son nombres de títulos válidos incluídos en el conjunto de títulos de la estructura. Ademas, el titulo que devuelve infoTitulo para cada clave en el diccionario con memoria sea el mismo titulo que figura en el conjunto de titulos.
5) La sumatoria de la cantidad de acciones de un determinado título entre todos los clientes de Wolfie

nunca supera el límite de "stock"del mismo.
6) Que las acciones totales de cada cliente que te da el diccionario inception sean las mismas que las que tiene cada infoPromesa ubicada en algun infoTitulo del diccionario con Memoria.
7) Que la cantidad de acciones disponibles sea la cantidad maxima del titulo menos las acciones que tienen los clientes.

5.5. Función de Abstracción

Abs: $\widehat{estr}$ e $\rightarrow$ Wolfie (Rep(e))
($\forall$ e: $\widehat{estr}$) Abs(e) $\equiv$w: $\widehat{Wolfie}$ /

$$\left(\begin{array}{l} clientes(w) =_{\text{obs}} e.conjuntoClientes \wedge \\ titulos(w) =_{\text{obs}} e.conjuntoTitulos \wedge \\ (\forall c : cliente)(c \in clientes(w)) \Rightarrow_{\text{L}} promesasDe(c, w) =_{\text{obs}} promesasDeDM(c, e.diccPromesas) \wedge \\ (\forall c : cliente)(c \in clientes(w)(((\forall nt : nombre)(\exists t : titulo)(t \in titulos(w))) \Rightarrow_{\text{L}} \\ accionesPorCliente(c, nt, w)) =_{\text{obs}} cantidad(e.diccClientesAcciones, c, nt) \end{array}\right)$$

5.6. Algoritmos

*i*INAUGURARWOLFIE(**in** *cs*: *conj*(*cliente*))$\rightarrow$ *resultado*: *wolfie*
1 *resultado.conjuntoTitulos* $\leftarrow$ *vacio*();
2 *resultado.conjuntoClientes* $\leftarrow$ *cs*;
3 *resultado.diccPromesas* $\leftarrow$ *vacioDM*();
4 *resultado.diccClientesAcciones* $\leftarrow$ *vacioDI*();
5 **var** *it* : *itConj*(*cliente*) = *crearIt*(*cs*);
6 **while** (*haySiguiente*(*it*))
7 *agregarClienteDI*(*resultado.diccClienteAcciones*, *siguiente*(*it*));
8 *avanzar*(*it*)
9 **end while**
10 **return** *resultado*
11 **end function**

Complejidad: $O(1)_1 + O(1)_2 + O(1)_{\text{vacioDM()}} + O(1)_{\text{vacioDI()}} + O(1)_6 + O(C)_7 * (O(C)_{\text{agregarClienteDI()}} + O(1)_{\text{avanzar}}) = O(C^2)$

*i*AGREGARTITULO(**inout** *e*: *estr*, **in** *t*: *titulo*)
1 **var** *it* : *itConj*(*titulo*)
2 *it* $\leftarrow$ *agregarRapido*(*e.conjuntoTitulos*, *t*);
3 *agregarTituloDM*(*e.diccPromesas*, *it*, *cardinal*(*e.conjuntoClientes*));
4 **end function**

Complejidad: $O(1)_{\text{agregarRapido}} + O(C + |nt|)_{\text{agregarTituloDM}} =$ **$O(C + |nt|)$**

*i*ACTUALIZARCOTIZACION(**in** *nt*: *nombre*, **in** *cot*: *Nat*, **inOut** *w*: *Wolfie*)
1 **var** *info* : *infoTitulo*
2 *info* $\leftarrow$ *obtenerDM*(*w.diccPromesas*, *nt*)
3 *recotizar*(*titulo*(*info*), *cot*)
4 **var** *it* : *itConj*(*infoPromesa*)//*iterador a conjunto de ventas*

```
5   it ← crearIt(ventas(info));
6   while (haySiguiente(it))
7         var sig : infoPromesa = siguiente(it)
8         if cot < limite(promesa(sig))
9           then
10                restarAcciones(w.diccClienteAcciones, cliente(sig), nombre(promesa(sig)), cantidad(promesa(sig)))
11                dispoibles(info) ← disponibles(info) + cantidad(promesa(sig));
12                eliminarSiguiente(it);
13        end if
14        avanzar(it);
15  end while
16  ordenar(compras(info));
17  var i : nat ← 1
18  while (i ≤ tamaño(compras(info)))
19        var comp : itConj(infoPromesa) ← dameIndice(i, compras(info));
20        if (comp ≠ NULL && cantidad(promesa(siguiente(comp))) ≤ disponibles(info))
21          then
22                sumarAcciones(w.diccClienteAcciones, cliente(siguiente(comp)),
23                  nombre(promesa(siguiente(comp))), cantidad(promesa(siguiente(comp)))
24                disponibles(info) ← disponibles(info) − cantidad(promesa(siguiente(comp)));
25                borrar(i, compras(info));
26        end if
27  end while
28  ordenar(compras(info));
29  return
30  end function
```

Complejidad:

$O(|nT|)_{obtenerDM}$ + $O(1)_{recotizar}$ + $O(1)_{crearIt}$ +
#Ventas *($O(\log C + |nT|)_{restarAcciones}$ + $O(1)_{11}$ + $O(1)_{eliminarSiguiente}$ + $O(1)_{avanzar}$) +
$O(\#compras * \log \#compras)_{ordenar}$ +
#compras *($O(1)_{dameIndice}$ + $O(1)_{20}$ + $O(\log C + |nT|)_{sumarAcciones}$ + $O(1)_{24}$ + $O(1)_{borrar}$) +
$O(\#compras * \log \#compras)_{ordenar}$ =

O(|nT|) + #Ventas *O(log C + |nT|) + O(#compras * log #compras) + #compras *O(log C + |nT|)
+ O(#compras * log #compras) =

Donde #compras y #ventas siempre va a ser menor o igual a la cantidad total de clientes (C) porque a lo sumo tienen una promesa de cada tipo.
⇒
O(|nT|) + C *O(log C + |nT|) + O(C * log C) + C *O(log C + |nT|) + O(C * log C) =
O(C*(log c + |nT|))

*i*AGREGARPROMESA(**in** c: *Cliente*, **in** p: *Promesa*, **inOut** w: *Wolfie*)

```
1   agregarPromesaDM(e.diccPromesas, p, c, &accionesTotales(e.diccClientesAcciones, c));
2   end function
```

Complejidad:

$O(|nT|)_{agregarPromesaDM}$ + $O(\log C)_{agregarPromesaDM}$ =
O(|nT| + log C)

iCLIENTES(**in** e: $estr$)$\rightarrow$ $resultado$: $conj(cliente)$
1 **return** $crearIt(e.conjuntoClientes)$
2 **end function**

Complejidad:
$\mathbf{O(1)}_{\text{crearIt}}$

iTITULOS(**in** e: $estr$)$\rightarrow$ $resultado$: $conj(titulo)$
1 **return** $crearIt(e.conjuntoTitulos)$
2 **end function**

Complejidad:
$\mathbf{O(1)}_{\text{crearIt}}$

iPROMESASDE(**in** e: $estr$, **in** c: $cliente$)$\rightarrow$ $resultado$: $conj(promesa)$
1 **return** $promesasDeDM(c, e.diccPromesas)$;
2 **end function**

Complejidad:
$\mathbf{O(T * C * |max\_nt|)}_{\text{promesasDeDM}}$

iACCIONESPORCLIENTE(**in** nt: $nombre$, **in** c: $cliente$, **in** e: $estr$)$\rightarrow$ $resultado$: Nat
1 **return** $cantidad(e.diccClientesAcciones, c, nt)$
2
3 **end function**

Complejidad:
$\mathbf{O(\log C + |nT|)}_{\text{cantidad}}$

5.7. Servicios usados

Diccionario Inception

Operacion	Aliasing	Complejidad
vacioDI	No	O(1)
AgregarClienteDI	No	O(C)
Definido?DI	No	O(log C)
ClientesDI	No	O(C)
obtenerDI	Si	O(log C)
AgregarTitulo	No	O(C + |nT|)
SumarAcciones	Si	O(log C + |nT|)
RestarAcciones	Si	O(log C + |nT|)
TitulosDe	Si	O(C)
Cantidad	Si	O(log C + |nT|)
AccionesTotales	Si	O(log C)

Diccionario Con Memoria

Operacion	Aliasing	Complejidad
vacioDM	No	O(1)
agregarTituloDM	No	O(num + |nT|)
definido?DM	No	O(|nT|)
obtenerDM	Si	O(|nT|)
titulosDM	Si	O(#claves)
agregarPromesaDM	No	O(|nT|)
promesasDeDM	Si	O(T * C * |max_nt|)

Usa el Modulo Conjunto Lineal de la Catedra

6. Módulo Diccionario Finito

6.1. Especificación

TAD Diccionario Finito(nat, significado)

igualdad observacional

$$(\forall d, d' : \text{dicc\_finito(nat, signficado)}) \left(d =_{\text{obs}} d' \iff \begin{array}{r} (tamMax(d) =_{\text{obs}} tamMax(d') \wedge tam(d) =_{\text{obs}} \\ tam(d') \wedge (\forall c : \text{nat})\ ((\text{def?}(c, d) =_{\text{obs}} \text{def?}(c, d')) \wedge_{\text{L}} \\ (\text{def?}(c, d) \Rightarrow_{\text{L}} \\ ((\text{obtener}(c, d) =_{\text{obs}} \text{obtener}(c, d')))))) \end{array} \right.$$

géneros dicc_finito(nat, significado)

exporta dicc_finito(nat, significado), generadores, observadores, $=_{\text{dicc}}$

usa BOOL, NAT, CONJUNTO(NAT)

observadores básicos

tamMax	: dicc_finito($nat \times significado$)	$\longrightarrow$	nat	
tam	: dicc_finito($nat \times significado$)	$\longrightarrow$	nat	
def?	: nat c $\times$ dicc_finito($nat \times significado$)	$\longrightarrow$	bool	
obtener	: nat c $\times$ dicc_finito($nat \times significado$)d	$\longrightarrow$	significado	(def?(c, d))

generadores

vacío	: nat $tamMax$	$\longrightarrow$	dicc_finito(nat, significado)
definir	: nat n $\times$ significado s $\times$ dicc_finito($nat \times sig$)d	$\longrightarrow$	dicc_finito(nat, significado) ($def?(n, d) \vee hayLugar(d)$)

otras operaciones

claves	: dicc_finito($nat \times significado$)d	$\longrightarrow$	conj(nat)
hayLugar	: dicc_finito($nat \times significado$)d	$\longrightarrow$	bool

axiomas $\forall d, e$: dicc _finito(nat, significado), $\forall c, k, n$: nat, $\forall s$: significado

tamMax(vacío(n))	$\equiv$	n
tamMax(definir(k, s, d))	$\equiv$	tamMax(d)
tam(vacío(n))	$\equiv$	0
tam(definir(k, s, d))	$\equiv$	**if** def?(k, d) **then** tam(d) **else** 1 + tam(d) **fi**
def?(c,vacío(n))	$\equiv$	*false*
def?(c, definir(k, s, d)	$\equiv$	$(c = k) \vee$ def?(c, d)
obtener(c, definir(k, s, d))	$\equiv$	**if** $c = k$ **then** s **else** obtener(c, d) **fi**
claves(vacío(n))	$\equiv$	$\emptyset$
claves(definir(c,s,d)	$\equiv$	Ag(c, claves(d))
hayLugar(d)	$\equiv$	$tam(d) < tamMax(d)$

Fin TAD

6.2. Servicios Exportados

Operacion	Aliasing	Complejidad
Vacio	No	O(n)
Definir	No	O(n * copiar info)
Definida?	No	O(log n)
Obtener	Si	O(log n)
Tamaño	No	O(1)
TamañoMax	No	O(1)
Claves	No	O(n)

n es la cantidad de claves

6.3. Interfaz

Interfaz DICC FINITO(NAT, SIGNIFICADO)
Se explica con: TAD Diccionario Finito(Nat, Significado)
Géneros: dicc_finito(nat,significado)
Operaciones:

Vacio(in n:nat) $\rightarrow$ *resultado*: dicc_finito(nat,significado)
Pre $\equiv\{\ true\ \}$
Post $\equiv\{\ resultado = vacio(n)\ \}$

Definir(inout d: dicc_ finito(nat,significado), in c: nat, in s:significado)
Pre $\equiv\{\ d = d_0\ \}$
Post $\equiv\{\ d = definir(c, s, d_0)\ \}$

Definida?(in c: nat, in d: dicc_ finito(nat, significado)) $\rightarrow$ *resultado*: bool
Pre $\equiv\{\ true\ \}$
Post $\equiv\{\ resultado = def?(c, d)\ \}$

Obtener(in c: nat, in d: dicc_ finito(nat,significado)) $\rightarrow$ *resultado*: significado
Pre $\equiv\{\ def?(c, d)\ \}$
Post $\equiv\{\ resultado = obtener(c, d)\ \}$

Claves(in d: dicc_ finito(nat,significado)) $\rightarrow$ *resultado*: conj(nat)
Pre $\equiv\{\ true\ \}$
Post $\equiv\{\ resultado = claves(d)\ \}$

Tamaño(in d: dicc_ finito(nat,significado)) $\rightarrow$ *resultado*: nat
Pre $\equiv\{\ true\ \}$

Post $\equiv\{\ resultado = tam(d)\ \}$

TamañoMax(in *d*: dicc_ finito(nat,significado)) $\rightarrow$ *resultado*: nat

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

Post $\equiv\{\ resultado = tamMax(d)\ \}$

6.4. Estructura de Representación

dicc_finito(nat,significado) se representa con arreglo_extendido_de_info

Donde arreglo_extendido_de_info es tupla$< tam : nat, arr : arreglo_dimensionable_de_info >$ info es tupla<clave: nat, sig:significado>

6.5. Invariante de Representación

Rep informal:
1 - Todos los naturales menores o iguales a e.tam están definidos, mientras que los mayores no.
2 - Para todos los nats definidos y distintos, sus claves son diferentes y respetan el orden de sus claves.

Rep: $\widehat{arreglo\_extendido\_de\_info} \rightarrow$ bool
($\forall$ e: $\widehat{arreglo\_extendido\_de\_info}$) Rep(e) = *true* $\Leftrightarrow$
($\forall$ i: nat)($(1 \leq i \leq$ e.tam $\wedge$ definido?(e.arr, i)) $\vee$ ($i >$e.tam $\wedge \neg$ definido?(e.arr, i)))
$\wedge$ ($\forall$ c1, c2: nat)((c1 $\neq$ c2 $\wedge$ definido?(e.arr, c1) $\wedge$ definido?(e.arr, c2))$\Rightarrow_{\text{L}}$(e.arr[c1].clave $\neq$ e.arr[c2].clave
$\wedge$ (c1 <c2 $\wedge$ e.arr[c1].clave <e.arr[c2].clave $\vee$ c2 <c1 $\wedge$ e.arr[c2].clave <e.arr[c1].clave))

6.6. Función de Abstracción

Abs: $\widehat{arreglo\_extendido\_de\_info}$ e $\rightarrow \widehat{dicc\_finito}$ (Rep(e))
($\forall$ arr: $\widehat{arreglo\_extendido\_de\_info}$)

$$\text{Abs(arr)} \equiv \text{d}: \widehat{dicc\_finito} \Leftrightarrow \left(\begin{array}{l} (\ e.tam =_{obs} tam(d) \wedge tamMax(d) =_{obs} tam(e.arr)\ ) \wedge \\ (\ (\forall c : nat) def?(c,d) \Rightarrow_{\text{L}} (\exists k : nat) definido?(k, e.arr) \wedge_{\text{L}}\ ) \\ (\ c = e.arr[k].clave \wedge obtener(c,d) =_{\text{obs}} e.arr[k].sig\ ) \end{array} \right)$$

6.7. Algoritmos

*i*Vacio(**in** n: *nat*)$\rightarrow$ *arreglo_extendido_de_info*
1 **return** $\langle 0, crearArreglo(n) \rangle$;
2 **end function**

Complejidad: $\text{O(n)}_{\text{crear arreglo}} + \text{O(2)}_{\text{crear tupla}} = \text{O(n)}$

```
iDefinir(inout d: arreglo_extendido_de_info, in k: info)
 1  if (e.tam < tam(d.arr))
 2     then d.arr[e.tam + 1] ← Copiar(k);
 3          var i: nat ← e.tam + 1
 4          while (i > 1 and e.arr[i].clave < e.arr[i − 1].clave)
 5               var temp: info ← Copiar(e.arr[i])
 6               e.arr[i] ← Copiar(e.arr[i − 1]);
 7               e.arr[i − 1] ← Copiar(temp);
 8               i − −;
 9          end while
10          d.tam + +;
11  end if
12  end function
```

Complejidad: $O(1)_1$ + $O(\text{copiar info})_2$ + $O(1)_3$ + $O(n)_4$ * ($O(\text{copiar info})_5$ + $O(\text{copiar info})_6$+ $O(\text{copiar info})_7$ + $O(1)_8$) + $O(1)_{10}$
= O(copiar info) + O(n) * O(3*copiar info)
= O(copiar info) + O(n) * O(copiar info)
= O(copiar info * (1 + n))
= O(n * copiar info)

```
iDefinida?(in c: clave, in d: arreglo_extendido_de_info)→ resultado: bool
 1  var i: nat ← 1
 2  var d: nat ← d.tam
 3  var m: nat
 4  var res: bool
 5  if (c < d.arr[i].clave OR c > d.arr[d].clave)
 6     then res ← false;
 7     else
 8          while (d > i + 1)
 9               m ← (i + d)/2;
10               if (c == d.arr[m].clave)
11                  then i ← m; d ← i;
12                  else if (c < d.arr[m].clave)
13                            then d ← m;
14                            else i ← m;
15                       end if
16               end if
17               res ← (d.arr[i] == c OR d.arr[d] == c);
18          end while
19  end if
20  return res;
21  end function
```

Complejidad: $O(1)_{1,\ 2,\ 3,\ 4,\ 5,\ 6}$ + $O(\log n)_{8,\ \text{por busqueda binaria}}$ * $O(1)_{9,\ 10,\ 11,\ 12,\ 13,\ 14,\ 15,\ 16,\ 17}$ + $O(1)_{20}$
= O(log n)

```
iObtener(in c: nat, in d: arreglo_extendido_de_info)→ resultado: significado
 1   //Genera Aliasing
 2  var i: nat ← 1
```

```
 3  var d: nat ← d.tam
 4  var m: nat
 5  while (d > i + 1)
 6        m ← (i + d)/2;
 7        if (c == d.arr[m].clave)
 8           then i ← m; d ← i;
 9           else if (c < d.arr[m].clave)
10                    then d ← m;
11                    else i ← m;
12                end if
13        end if
14  end while
15  return d.arr[m].sig;
16  end function
```

Complejidad: $O(1)_{2,\ 3,\ 4}$ + $O(\log n)_{5,\ \text{por busqueda binaria}}$ * $O(1)_{6,\ 7,\ 8,\ 9,\ 10,\ 11}$ + $O(1)_{16}$
= O(log n)

```
iClaves(in d: arreglo_extendido_de_info)→ resultado: conj(nat)
 1  var i: nat
 2  resultado = Vacio();
 3  for (i ← 1 to e.tam)
 4        AgregarRpido(d.arr[i].clave, resultado);
 5  end for
 6  return resultado;
 7  end function
```

Complejidad: $O(1)_{1,\ 2}$ + $O(n)_3$ * $O(1)_4$ + $O(1)_6$
= O(n)

```
iTamaño(in d: arreglo_extendido_de_info)→ resultado: nat
 1  return e.tam;
 2  end function
```

Complejidad: O(1)

```
iTamañoMax(in d: arreglo_extendido_de_info)→ resultado: nat
 1  return tam(d.arr);
 2  end function
```

Complejidad: O(1)

7. Módulo Diccionario String

7.1. Servicios Exportados

Operacion	Aliasing	Complejidad
CrearDiccString	No	O(1)
DefinirDS	No	O(|c|)
Definido?DS	No	O(|c|)
ObtenerDS	Si	O(|c|)
ClavesDS	Si	O(N)
CopiarDS	No	O(N*(|c_{max}|+O(copiarSignificado)))

N es la cantidad de claves.
c es la clave, |c| es su largo.

Asumimos:
Todos los arreglos empiezan en 1
Para los strings asumimos que a cada char se puede acceder en O(1)
y que se puede obtener la longitud en O(N) y su creacion es proporcional a la longitud.
Asumimos que la posicion i del array corresponde a la letra i del abecedario.
Longitud: long(string)
Acceso: String[nat]
int(char) nos da el numero entre 1 y 27 que le corresponde en el abecedario.
TAD Clave es String

7.2. Interfaz

Interfaz DICCIONARIO STRING(CLAVE, SIGNIFICADO)
Se explica con: Diccionario(Clave, Significado)
Géneros: diccionarioDS
Operaciones:

CrearDiccString() $\rightarrow$ *resultado*: diccionarioDS

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

Post $\equiv\{$ $resultado = vacio$ $\}$

DefinirDS(**inout** d: $diccionarioDS$, **in** c: $clave$, **in** s: $significado$)

Pre $\equiv\{$ $d = d_0$ $\}$

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

Definido?DS(**inout** d: $diccionarioDS$, **in** c: $clave$) $\rightarrow$ *resultado*: bool

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

Post $\equiv\{$ $resultado = def?(c, d)$ $\}$

$ObtenerDS(\textbf{inout}\, d\text{: } diccionarioDS, \textbf{in}\, c\text{: } clave) \rightarrow resultado\text{: significado}$

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

Post $\equiv\{\ resultado = obtener(c,d)\ \}$

$ClavesDS(\textbf{inout}\, d\text{: } diccionarioDS) \rightarrow resultado\text{: conj(clave)}$

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

Post $\equiv\{\ resultado = claves(d)\ \}$

$CopiarDS(\textbf{inout}\, d\text{: } diccionarioDS) \rightarrow resultado\text{: diccionarioDS}$

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

Post $\equiv\{\ resultado = d\ \}$

7.3. Estructura de Representación

diccionario se representa con estr
Donde estr es tupla < definiciones: nodo , claves: conj(string), conjSignificados: conj(significado) >
Donde nodo es tupla <siguientes: arreglo[27] de puntero a nodo, significado: puntero a significado >

7.4. Invariante de Representación

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

(∀ d: $\widehat{estr}$) Rep(d) ⇔
$$\left(\begin{array}{l} \text{No hay ciclos en d.definiciones} \\ \text{Todos los caminos con significados validos estan en claves} \\ \text{Para todo elemento del conjunto claves existe un camino de nodos valido} \\ \text{ que tiene un significado valido} \\ \text{No hay caminos finales (con todos los siguientes NULL) que no tengan significado valido} \\ \text{Todos los significados validos apuntan a algun elemento de e.conjSignificados} \\ \text{ y todos los elementos de e.conjSignificados tienen algun significado valido} \\ \text{ con un puntero hacia el.} \end{array}\right)$$

7.5. Función de Abstracción

Abs: $\widehat{estr}$ e → diccionario (Rep(d))
(∀ e: $\widehat{estr}$) Abs(e) ≡ d: $\widehat{diccionario}$ /
$$\left(\begin{array}{l} (\forall c : clave) def?(c,d) \Leftrightarrow \text{(la clave tiene su camino valido con significado valido)} \wedge_{\text{L}} \\ (\forall c : clave) def?(c,d) \Rightarrow_{\text{L}} obtener(c,d) =_{\text{obs}} \text{(el significado en e.conjSignificados al cual apunta} \\ \text{ el puntero del camino de c)} \end{array}\right)$$

7.6. Algoritmos

iCrearDiccString()→ $resultado$: $estr$

```
1  var i : nat
2  resultado.claves = Vacio()
3  for i ← 1 to 27
4        resultado.definiciones.siguientes[i] = NULL
5  end for
6  resultado.definiciones.significado = NULL
7  return resultado
8  end function
```

Complejidad: 27 * O(1)=**O(1)**
Aliasing: No

```
iDefinirDS(inout d: estr, in c: clave, in s: significado)
 1  Agregar(c, d.claves);
 2  Agregar(s, d.conjSiguientes);
 3  var punt: punteroAsignificado ← &s
 4  var actual: nodo
 5  actual = d.definiciones;
 6  for i = 1 to Longitud(c)
 7        if actual.siguientes[int(c[i])] == NULL
 8           then
 9                 var nuevoNodo: nodo
10                 nuevoNodo.significado = NULL
11                 for j = 1 to 27
12                       nuevoNodo.siguientes[j] = NULL
13                 end for
14                 actual.siguientes[c[i]] = &nuevoNodo
15        end if
16        actual = (*actual).siguientes[int(c[i])];
17  end for
18  actual.significado = *punt
19  end function
```

Complejidad: |c| *(27*(O(1))) = O(|c|)
Aliasing: No

```
iDefinido?DS(in d: estr, in c: clave)→ resultado: bool
 1  var def: bool
 2  var actual: nodo
 3  if (d.definiciones.siguientes[c[1]]) ≠ NULL
 4     then
 5           actual = *(d.definiciones.siguientes[c[1]])
 6           def ← true
 7           for i = 2 to Longitud(c)
 8                 if (actual.siguientes[int(c[i])] ≠ NULL && def)
 9                    then
10                          actual = *(actual.siguientes[int(c[i])])
11                    else
12                         def ← false
13                 end if
14           end for
```

```
15            if actual.significado ≠ NULL
16              then
17                    return def
18              else
19                   def ← false
20            end if
21      else
22           def ← false
23    end if
24    return def
25    end function
```

Complejidad: **O(|c|)**
Aliasing: No

*i*ObtenerDS(**in** *d*: *estr*, **in** *c*: *clave*)→ *resultado*: *significado*

```
1   var actual: nodo
2   actual = d.definiciones
3   for i = 1 to Longitud(c)
4         actual = *(actual.siguientes[int(c[i])])
5   end for
6   return actual.significado
7   end function
```

Complejidad: **O(|c|)**
Aliasing: Si

*i*ClavesDS(**in** *d*: *estr*)→ *resultado*: *conj*

```
1   return d.claves
2   end function
```

Complejidad: **O(1)**
Aliasing: Si

*i*CopiarDS(**in** *d*: *diccionario*)→ *resultado*: *diccionario*

```
1   resultado.claves = Copiar(d.claves)
2   resultado.conjSignificados = Copiar(d.conjSignificados)
3   for i = 1 to 27
4         if d.siguientes[i]! = NULL
5            then resultado.siguientes[i] = *iCopiarNodo(d.siguientes[i])
6            else resultado.siguientes[i] = NULL
7         end if
8   end for
9   return resultado
10  end function
```

Complejidad: O(#(claves)*|$\text{clave}_{\max}$|)+O(#(significados)*copiarSignificado)+27*O(copiarNodo)
Donde #(claves) == #(significados)
O(copiarNodo)=O(copiarSignificado) ⇒
O(#(claves) (O(|$\text{clave}_{\max}$|)+ O(copiarSignificado)))+O(copiarSignificado) =
O(#(claves) (O(|$\text{clave}_{\max}$|)+ O(copiarSignificado)))

Aliasing: No

```
iCopiarNodo(in n: puntero(nodo))→ resultado: nodo
 1  if &n.significado! = NULL
 2     then resultado.significado ← Copiar(&n.significado)
 3     else resultado.significado = NULL
 4  end if
 5  for i = 1 to 27
 6        if &n.siguientes[i]! = NULL
 7           then resultado.siguientes[i] = *iCopiarNodo(n.siguientes[i])
 8           else resultado.siguientes[i] = NULL
 9        end if
10  end for
11  return resultado
12  end function
```

Complejidad: 27*O(copiarSignificado) = **O(copiarSignificado)**
Aliasing: No

8. Módulo Diccionario Inception

8.1. Especificación

TAD DICCIONARIO INCEPTION

Extiende al TAD Diccionario Finito.
TAD Diccionario Finito es: dicc(cliente, diccionarioDS(nombre, nat))
Donde diccionarioDS es un renombre de diccionario.

Generos: diccIncep

Donde las siguientes operaciones son renombres de:

DiccIncep	**DiccFinito**
vacioDI()	Vacio()
Definido?DI(d,c)	Def?(c,d)
ClientesDI(d)	Claves(d)
ObtenerDI(d, c)	Obtener(c,d)
AgregarClienteDI(d, c)	Definir(c, crearDiccString(), d)

EXPORTA:

otras operaciones

agregarTitulo : d *diccIncep* × c *cliente* × nT *nombre* × n *nat* $\longrightarrow$ dicc {c ∈clientesDI(d)}

sumarAcciones : d *diccIncep* × c *cliente* × nT *nombre* × n *nat* $\longrightarrow$ dicc
{c ∈ clientesDI(d) $\wedge_{\text{L}}$nT ∈ titulosDe(d, c)}

restarAcciones : d *diccIncep* × c *cliente* × nT *nombre* × n *nat* $\longrightarrow$ dicc
{c ≤ cantidad(d, c, nT) ∧ c ∈ clientesDI(d) $\wedge_{\text{L}}$nT ∈ titulosDe(d, c)}

titulosDe : d *diccIncep* × c *cliente* $\longrightarrow$ conj(nombre) {c ∈clientesDI(d)}

cantidad : d *diccIncep* × c *cliente* × nT *nombre* $\longrightarrow$ nat
{c ∈ clientesDI(d) $\wedge_{\text{L}}$nT ∈ titulosDe(d, c)}

accionesTotales : d *diccIncep* × c *cliente* $\longrightarrow$ nat {c ∈ clientesDI(d)}

sumatoriaDeAcciones : d *diccIncep* × c *cliente* × conj *conj*(*nombre*) $\longrightarrow$ nat
{c ∈ clientesDI(d) $\wedge_{\text{L}}$nT ∈ titulosDe(d, c)}

axiomas ∀*d*: diccIncep, ∀*c*: cliente, ∀*n*: nat, ∀*conj*(*nombre*): conj

agregarTitulo(d, c, nT, n) ≡ definirDS(obtener(d,c), nT, n)

sumarAcciones(d, c, nT, n) ≡ definirDS(obtener(d,c), nT, cantidad(d, c, nT)+n)

restarAcciones(d, c, nT, n) ≡ definirDS(obtener(d,c), nT, cantidad(d, c, nT)-n)

titulosDe(d, c) ≡ clavesDS(obtener(d,c))

cantidad(d, c, nT) ≡ obtenerDS(obtener(d,c))

accionesTotales(d, c ≡ sumatoriadeAcciones(d, c, titulosDe(d, c))

sumatoriaDeAcciones(d, c, ∅) ≡ 0

sumatoriaDeAcciones(d, c, Ag(nT, conj)) $\equiv$ cantidad(d, c, nT) + sumatoriaDeAcciones(d, c, conj)

Fin TAD

8.2. Servicios Exportados

Operacion	Aliasing	Complejidad
vacioDI	No	O(1)
AgregarClienteDI	No	O(C)
Definido?DI	No	O(log C)
ClientesDI	No	O(C)
obtenerDI	Si	O(log C)
AgregarTitulo	No	O(C + |nT|)
SumarAcciones	Si	O(log C + |nT|)
RestarAcciones	Si	O(log C + |nT|)
TitulosDe	Si	O(C)
Cantidad	Si	O(log C + |nT|)
AccionesTotales	Si	O(log C)

8.3. Interfaz

Interfaz DICCIONARIO INCEPTION
Se explica con: TAD Diccionario Inception
Que es dicc(cliente, diccionarioDS(nombre, nat))
Donde TAD dicc es Diccionario Finito y diccionarioDS es un renombre de diccionario.
Se representa extendiendo: dicc(cliente, diccionarioDS(nombre, nat))
Generos: dIncep
Operaciones:

VacioDI() $\rightarrow$ *resultado*: dIncep
Pre $\equiv \{ \ true \ \}$
Post $\equiv \{ \ resultado =_{\text{obs}} vacio \ \}$

AgregarClienteDI(**inout** *dicc*: *dIncep*, **in** *clave*: *cliente*) $\rightarrow$ *resultado*: dIncep
Pre $\equiv \{ \ dicc =_{\text{obs}} dicc_0 \wedge clave \notin claves(dicc_0) \ \}$
Post $\equiv \{ \ resultado =_{\text{obs}} AgregarClienteDI(dicc_0, clave) \ \}$

Definido?DI(**in** *dicc*: *dIncep*, **in** *clave*: *cliente*) $\rightarrow$ *resultado*: bool
Pre $\equiv \{ \ dicc =_{\text{obs}} dicc_0 \ \}$
Post$\equiv \{ \ resultado =_{\text{obs}} definido?DI(clave, dicc_0) \ \}$

ClientesDI(**in** *dicc*: *dIncep*) $\rightarrow$ *resultado*: conj(cliente)
Pre $\equiv \{ \ dicc =_{\text{obs}} dicc_0 \ \}$

Post $\equiv \{\ resultado =_{\text{obs}} ClientesDI(dicc_0)\ \}$

$ObtenerDI$(**in** $dicc$: $dIncep$, **in** $clave$: $cliente$) $\rightarrow$ *resultado*: diccionarioDS(nombre, nat)

Pre $\equiv \{\ dicc =_{\text{obs}} dicc_0 \wedge def?(clave, dicc) \equiv true\ \}$

Post $\equiv \{\ resultado =_{\text{obs}} obtenerDI(dicc_0, clave)\ \}$

$AgregarTitulo$(**inout** $dicc$: $dIncep$, **in** $clave$: $cliente$, **in** nT: $nombre$, **in** c: nat) $\rightarrow$ *resultado*: dIncep

Pre $\equiv \{\ dicc =_{\text{obs}} dicc_0 \wedge def?(nT, obtener(clave, dicc).diccionario) \equiv false\ \}$

Post $\equiv \{\ resultado =_{\text{obs}} agregarTitulo(dicc_0, clave, nT, c)\ \}$

$SumarAcciones$(**inout** $dicc$: $dIncep$, **in** $clave$: $cliente$, **in** nT: $nombre$, **in** c: nat) $\rightarrow$ *resultado*: dIncep

Pre $\equiv \left\{ \begin{array}{l} dicc =_{\text{obs}} dicc_0 \wedge \\ def?(clave, dicc) \equiv true \wedge \\ (nT \in claves(obtener(clave, dicc).diccionario \equiv true) \end{array} \right\}$

Post $\equiv \{\ resultado =_{\text{obs}} sumarAcciones(dicc_0, clave, nT, c)\ \}$

$RestarAcciones$(**inout** $dicc$: $dIncep$, **in** $clave$: $cliente$, **in** nT: $nombre$, **in** c: nat) $\rightarrow$ *resultado*: dIncep

Pre $\equiv \left\{ \begin{array}{l} dicc =_{\text{obs}} dicc_0 \wedge \\ def?(clave, dicc) \equiv true \wedge \\ (nT \in claves(obtener(clave, dicc) \equiv true) \end{array} \right\}$

Post $\equiv \{\ resultado =_{\text{obs}} restarAcciones(dicc_0, clave, nT, c)\ \}$

$TitulosDe$(**in** $dicc$: $dIncep$, **in** $clave$: $cliente$) $\rightarrow$ *resultado*: conj(Nombre)

Pre $\equiv \{\ dicc =_{\text{obs}} dicc_0 \wedge definido?DI(dicc, clave)\ \}$

Post $\equiv \{\ resultado =_{\text{obs}} TitulosDe(dicc_0, clave)\ \}$

$Cantidad$(**in** $dicc$: $dIncep$, **in** $clave$: $cliente$, **in** nT: $nombre$) $\rightarrow$ *resultado*: nat

Pre $\equiv \{\ dicc =_{\text{obs}} dicc_0\ \}$

Post $\equiv \{\ resultado =_{\text{obs}} Cantidad(dicc_0, clave, nT)\ \}$

$accionesTotales$(**in** e: $estr$, **in** c: $cliente$) $\rightarrow$ *resultado*: nat

Pre $\equiv \{\ dicc =_{\text{obs}} dicc_0\ \}$

Post $\equiv \{\ resultado =_{\text{obs}} accionesTotales(dicc_0, c)\ \}$

8.4. Estructura de Representación

diccIncep(α) se representa con estr_diccIncep

Donde estr_diccIncep es dicc_Finito(cliente, tupla<titulos: diccionarioDS(nombre, nat), accionesTotales: nat >)

8.5. Invariante de Representación

Vale el invariante del Tad Diccionario Finito, y ademas:
• Las Acciones Totales de cada cliente es la suma de las acciones de cada titulo.

Rep: $\widehat{diccIncep} \rightarrow$ bool
($\forall$ e: $\widehat{diccIncep}$) Rep(dicc) = *true* $\Leftrightarrow$
Rep(DiccionarioFinito) $\wedge$
($\forall$ c: cliente) definido?DI(e, c) $\Rightarrow_{\text{L}}$($\forall$ nT: nombre) nT $\in$ TitulosDe(e, c) $\Rightarrow_{\text{L}}$
ObtenerDS(Obtener(e, c), nT).accionesTotales == sumatoriaDeAcciones(e, c, titulosDe(e, c))

8.6. Función de Abstracción

Vale la funcion de Abstraccion del Modulo Diccionario Finito. Ademas

Abs: $\widehat{estr}$ e $\rightarrow$ diccIncep (Rep(e))
($\forall$ e: $\widehat{estr}$) Abs(e) $\equiv$ d: $\widehat{diccIncep}$ /

$$\left(\begin{array}{l} ((\forall c : cliente)c \in clientesDI(d) \Rightarrow_{\text{L}} \\ titulosDe(d,c) =_{\text{obs}} titulosDe(e,c) \wedge accionesTotales(d,c) == accionesTotales(e,c) \wedge \\ (\forall nT : nombre, \forall n : nat)agregarTitulo(d,c,nT,n) =_{\text{obs}} agregarTitulo(e,c,nT,n) \wedge \\ (\forall nT : nombre)nT \in clavesDS(obtener(c,d)) \Rightarrow_{\text{L}} \\ sumarAcciones(d,c,nT,c) =_{\text{obs}} sumarAcciones(e,c,nT,c) \wedge \\ restarAcciones(d,c,nT,c) =_{\text{obs}} restarAcciones(e,c,nT,c) \wedge \\ cantidad(d,c,nT) = cantidad(e,c,nT) \end{array}\right)$$

8.7. Algoritmos

En la Complejidad: C es la cantidad de clientes en el Wolfie, nT el nombre del titulo

*i*VacioDI()$\rightarrow$ *resultado*: *estr*
1 **return** *Vacio*()
2 **end function**

Complejidad: **O(1)** $_{\text{Vacio()}}$
Aliasing: No

*i*AgregarClienteDI(**inout** *e*: *estr*, **in** *c*: *cliente*)
1 *definir*(*c*, *vacioDS*(), *e*)
2 *obtener*(*e*, *c*).*accionesTotales* $\leftarrow$ 0
3
4 **end function**

Complejidad: O(C*copiarInfo)$_{\text{definir}}$ + O(log C) $_{\text{obtener}}$
Donde O(copiarInfo)=O(vacioDS)=O(1)
$\Rightarrow$ O(C * 1 + log C)=**O(C)**
Aliasing: No

*i*DEFINIDO?DI(**in** e: *estr*, **in** c: *cliente*)$\rightarrow$ *resultado*: *bool*
1 **return** $definida?(c, e)$
2 **end function**

Complejidad: **O(log C)** $_{\text{definida?}}$
Aliasing: No

*i*CLIENTESDI(**in** e: *estr*)$\rightarrow$ *resultado*: $conj(cliente)$
1 **return** $claves(e)$
2 **end function**

Complejidad: **O(C)** $_{\text{claves}}$
Aliasing: No

*i*OBTENERDI(**in** e: *estr*, **in** c: *cliente*)$\rightarrow$ *resultado*: $diccionarioDS(nombre, nat)$
1 **return** $obtener(c, e)$
2 **end function**

Complejidad: **O(log C)** $_{\text{obtener}}$
Aliasing: Si

*i*AGREGARTITULO(**inout** e: *estr*, **in** c: *cliente*, **in** nT: *nombre*, **in** n: *nat*)
1 **var** *punt*: $punteroAdiccionarioDS$
2 $punt \leftarrow \&obtener(e, c)$
3 $(*punt).definirDS(nT, n)$
4 **end function**

Complejidad: O(log C)$_{\text{obtener}}$+O(|nT|)$_{\text{definirDS}}$ = **O(C + |nT|)**
Aliasing: No

*i*SUMARACCIONES(**inout** e: *estr*, **in** c: *cliente*, **in** nT: *nombre*, **in** n: *nat*)
1 **var** *tenia*: *nat*
2 $tenia \leftarrow cantidad(e, c, nT)$
3 **var** *punt*: $punteroAdiccionarioDS$
4 $punt \leftarrow \&obtener(e, c)$
5 $(*punt).definirDS(nT, tenia + n)$
6 $obtener(e, c).accionesTotales \leftarrow obtener(e, c).accionesTotales + n$
7 **end function**

Complejidad: O(|nT| + log C) $_{\text{cantidad}}$ + O(log C)$_{\text{obtener}}$+O(|nT|)$_{\text{definirDS}}$ +
O(log C) $_{\text{obtener}}$ + O(log C) $_{\text{obtener}}$=
O(|nT| + log C + log C + |nT| + log C + log C) = **O(|nT| + log C)**
Aliasing: Si

*i*RESTARACCIONES(**inout** e: *estr*, **in** c: *cliente*, **in** nT: *nombre*, **in** n: *nat*)
1 **var** *tenia*: *nat*
2 $tenia \leftarrow cantidad(e, c, nT)$

3 **var** *punt*: *punteroAdiccionarioDS*
4 *punt* ← &*obtener*(e, c)
5 $(*punt).definirDS(nT, tenia - n)$
6 $obtener(e, c).accionesTotales \leftarrow obtener(e, c).accionesTotales - n$
7 **end function**

Complejidad: O(|nT| + log C) $_{\text{cantidad}}$ + O(log C)$_{\text{obtener}}$ +O(|nT|)$_{\text{definirDS}}$ + O(log C) $_{\text{obtener}}$ + O(log C) $_{\text{obtener}}$=
O(|nT| + log C + log C + |nT| + log C + log C) = **O(|nT| + log C)**
Aliasing: Si

*i*TITULOSDE(**in** e: *estr*, **in** c: *cliente*)→ *resultado*: *conj*(*nombre*)
1 **return** *clavesDS*(*obtener*(d, c))
2 **end function**

Complejidad: O(C)$_{\text{clavesDS}}$ + O(log C)$_{\text{obtener}}$ =
O(C + log C) = **O(C)**
Aliasing: Si

*i*CANTIDAD(**in** e: *estr*, **in** c: *cliente*, **in** nT: *nombre*)→ *resultado*: *nat*
1 **var** *cant*: *nat*
2 **if** $definido?DS(obtener(e, c), nT)$
3 **then**
4 $cant \leftarrow obtenerDS(obtener(e, c), nT)$
5 **else**
6 $cant \leftarrow 0$
7 **end if**
8 **return** *cant*
9 **end function**

Complejidad: O(|nT|) $_{\text{definido?DS}}$ + O(log C) $_{\text{obtener}}$ + O(|nT|)$_{\text{obtenerDS}}$ + O(log C) $_{\text{obtener}}$=
O(|nT| + log C)
Aliasing: Si

*i*ACCIONESTOTALES(**in** e: *estr*, **in** c: *cliente*)→ *resultado*: *nat*
1 **return** $obtener(e, c).accionesTotales$
2 **end function**

Complejidad: **O(log C)** $_{\text{obtener}}$
Aliasing: Si

8.8. Servicios usados

Diccionario Finito

Operacion	Aliasing	Complejidad
Vacio	No	O(n)
Definir	No	O(n * copiar info)
Definida?	No	O(log n)
Obtener	Si	O(log n)
Tamaño	No	O(1)
TamañoMax	No	O(1)
Claves	No	O(n)

n es la cantidad de claves

Diccionario String

Operacion	Aliasing	Complejidad
CrearDiccString	No	O(1)
DefinirDS	No	O(|c|)
Definido?DS	No	O(|c|)
ObtenerDS	Si	O(|c|)
ClavesDS	Si	O(N)
CopiarDS	No	O(N*(|c_{max}|+O(copiarSignificado)))

N es la cantidad de claves.
c es la clave, |c| es su largo.

Usa el Modulo Conjunto Lineal de La Catedra.

9. Módulo Diccionario con memoria

9.1. Especificación

TAD DICCIONARIO CON MEMORIA

Extiende al TAD Diccionario de la Catedra.

Donde las siguientes operaciones son renombres de:

DiccionarioConMemoria	DiccionarioDeLaCatedra
vacioDM()	Vacio()
obtenerDM(d, t)	obtener(t,d)
definido?DM(t,d)	definido?(t,d)
titulosDM(d)	claves(d)
agregarTituloDM(d, t, n)	definir(d, nombre(t), nuevoInfoTitulo(t, $\emptyset$, $crear(n)$, #maxAcciones(t)))

géneros dMemory

exporta generadores, observadores, otras operaciones

usa DICCIONARIO, BOOL, NAT

otras operaciones

agregarPromesaDM : dMemory d × promesa pr × cliente c × nat n $\longrightarrow$ dMemory
{ titulo(pr) ∈ titulosDM(d) $\wedge_{\text{L}}$((tipo(pr)==compra ∧ pr ∉ compras(obtener(d, titulo(pr)))) ∨ (tipo(pr)==venta ∧ pr ∉ ventas(obtener(d, titulo(pr)))))}

promesasDeDM : dMemory d × cliente c $\longrightarrow$ conj(promesa)

unir : cliente c × conj(nombre) ts × dMemory d $\longrightarrow$ conj(promesa)

filtrar : cliente c × conj(infProm) iPs $\longrightarrow$ conj(promesa)

axiomas ∀d: dMemory, ∀t: titulo, ∀pr: promesa, ∀c: cliente, ∀n: nat, ∀ts: conj(nombre), ∀iPs: conj(infProm)

agregarPromesaDM(d, pr, c, n) ≡ **if** tipo(pr) = Compra **then**
definir(d, titulo(pr), nuevoInfoTitulo(
titulo(obtenerDM(d, titulo(pr))), ventas(obtenerDM(d, titulo(pr))),
insertar(nuevoInfoProm(c, pr, n), compras(obtenerDM(d, titulo(pr)))),
disponibles(obtenerDM(d, titulo(pr)))))
else
definir(d, titulo(pr), nuevoInfoTitulo(
titulo(obtenerDM(d, titulo(pr))), ventas(obtenerDM(d, titulo(pr))) U {nuevoInfoProm(c, pr, n)},
compras(obtenerDM(d, titulo(pr))), disponibles(obtenerDM(d, titulo(pr)))))
fi

promesasDeDM(d, c) ≡ unir(c, titulosDM, d)

unir(c, $\emptyset$, d) $\equiv$ $\emptyset$

unir(c, ts, d) $\equiv$ filtrar(c, sinUno(ts), d)$\cup$ filtrar(c, ventas(obtenerDM(d, dameUno(ts))))$\cup$ filtrar(c, compras(obtenerDM(d, dameUno(ts)))

filtrar(c, $\emptyset$) $\equiv$ $\emptyset$

filtrar(c, iPs) $\equiv$ **if** cliente(dameUno(iPs))$=_{\text{obs}}$ c **then** promesa(dameUno(iPs)) **else** $\emptyset$ **fi** $\cup$ filtrar(c, sinUno(iPs))

Fin TAD

9.2. Servicios Exportados

Operacion	Aliasing	Complejidad
vacioDM	No	O(1)
agregarTituloDM	No	O(num + |nT|)
definido?DM	No	O(|nT|)
obtenerDM	Si	O(|nT|)
titulosDM	Si	O(#claves)
agregarPromesaDM	No	O(|nT|)
promesasDeDM	Si	O(T * C * |max_nt|)

9.3. Interfaz

Interfaz DICCIONARIO MEMORIA
Se explica con TAD Diccionario con Memoria
G'eneros: dMemory
Operaciones:

$VacioDM()$ $\rightarrow$ *resultado*: dMemory

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

Post $\equiv$ $\{$ $resultado =_{\text{obs}} vacioDM()$ $\}$

$agregarTituloDM$(**inout** $dicc$: $dMemory$, **in** t: $titulo$, **in** n: nat)

Pre $\equiv$ $\{$ $dicc =_{\text{obs}} dicc_0 \wedge titulo \notin claves(dicc_0)$ $\}$

Post $\equiv$ $\{$ $dicc = agregarTituloDM(dicc_0, t, n)$ $\}$

$definido?DM$(**in** $dicc$: $dMemory$, **in** nt: $nombre$) $\rightarrow$ *resultado*: bool

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

Post$\equiv$ $\{$ $resultado =_{\text{obs}} definido?DM(nt, dicc)$ $\}$

$obtenerDM$(**in** $dicc$: $dMemory$, **in** nt: $nombre$) $\rightarrow$ *resultado*: infoTitulo

Pre $\equiv$ $\{$ $definido?DM(nt, dicc)$ $\}$

Post $\equiv \{\ resultado =_{\text{obs}} obtenerDM(dicc, nt)\ \}$

$titulosDM(\textbf{in}\, dicc\colon dMemory) \rightarrow resultado\colon$ conj(nombre)

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

Post $\equiv \{\ resultado =_{\text{obs}} titulosDM(dicc)\ \}$

$agregarPromesaDM(\textbf{inout}\, dicc\colon dMemory, \textbf{in}\, pr\colon promesa, \textbf{in}\, cl\colon cliente, \textbf{in}\, pN\colon punteroANat)$

Pre $\equiv \{\ dicc =_{\text{obs}} dicc_0 \wedge titulo(pr) \in titulosDM(dicc_0)\ \}$

Post $\equiv$
$$\left\{ \begin{array}{l} TitulosDM(dicc_0) =_{\text{obs}} TitulosDM(dicc) \wedge_{\text{L}} \\ ((\forall nT)(nT \in TitulosDM(dicc) \wedge nT \neq titulo(pr)) \Rightarrow_{\text{L}} obtenerDM(dicc_0, nT) =_{\text{obs}} obtenerDM(dicc, nT)) \wedge \\ titulo(obtenerDM(dicc, titulo(pr))) =_{\text{obs}} titulo(obtenerDM(dicc_0, titulo(pr))) \wedge \\ disponibles(obtenerDM(dicc, titulo(pr))) =_{\text{obs}} disponibles(obtenerDM(dicc_0, titulo(pr))) \wedge \\ ((tipo(pr) == compra \wedge \\ (compras(obtenerDM(dicc, titulo(pr))) =_{\text{obs}} \\ compras(obtenerDM(dicc, titulo(pr))) \cup \{nuevoInfoPromesa(cl, pr, pN)\}) \wedge \\ ventas(obtenerDM(dicc, titulo(pr))) =_{\text{obs}} ventas(obtenerDM(dicc_0, titulo(pr)))) \vee \\ (tipo(pr) == venta \wedge \\ (ventas(obtenerDM(dicc, titulo(pr))) =_{\text{obs}} \\ ventas(obtenerDM(dicc, titulo(pr))) \cup \{nuevoInfoPromesa(cl, pr, pN)\}) \wedge \\ compras(obtenerDM(dicc, titulo(pr))) =_{\text{obs}} compras(obtenerDM(dicc_0, titulo(pr))))) \end{array} \right\}$$

$promesasDeDM(\textbf{in}\, dicc\colon dMemory, \textbf{in}\, cl\colon cliente) \rightarrow resultado\colon$ conj(promesa)

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

Post $\equiv$
$$\left\{ \begin{array}{l} \forall nT \in titulosDM(dicc) \Rightarrow_{\text{L}} \\ (\forall infT \in obtener(nT, dicc) \Rightarrow_{\text{L}} \\ (\forall infPV \in ventas(infT) \Rightarrow_{\text{L}} (cliente(infPV) = cl \Leftrightarrow promesa(infPV) \in resultado) \\ (\forall infPC \in compras(infT) \Rightarrow_{\text{L}} (cliente(infPC) = cl \Leftrightarrow promesa(infPC) \in resultado) \end{array} \right\}$$

9.4. Estructura de Representación

diccMemory(α) se representa con estr_diccMemory
Donde estr_diccMemory es tupla<
fueModificado: bool,
cliente: nat,
ultimasPromesas: conj(promesa),
diccPorTitulo: diccString(nombre, infoTitulo)
>

9.5. Invariante de Representación

- Todas las ultimas promesas son promesas que pertenecen al significado de algun titulo.
- Cada clave tiene como significado un infoTitulo cuyo nombre de titulo es ella misma.

Rep: $\widehat{dMemory} \to$ bool
(∀ e: $\widehat{dMemory}$) Rep(dicc) = $true \Leftrightarrow$
$((\forall p : promesa)p \in e.ultimasPromesas \Rightarrow$
$(\exists nT : nombre)nT \in titulosDM(dicc) \Rightarrow_{\text{L}}$
$((tipo(p) == venta \wedge$
$(\exists infoP : infoProm)(infoP \in ventas(obtener(e.diccPorTitulo, nT)) \wedge$
$p =_{\text{obs}} promesa(infoP))) \vee$
$(tipo(p) == compra \wedge$
$(\exists infoP : infoProm)(infoP \in compras(obtener(e.diccPorTitulo, nT)) \wedge$
$p =_{\text{obs}} promesa(infoP))))) \wedge$
$(\forall t : nombre)t \in titulosDM(dicc) \Rightarrow_{\text{L}}$
$nombre(titulo(obtener(dicc, t))) = t$

9.6. Función de Abstracción

Abs: $\widehat{dMemory}$ e $\to$ $\widehat{dMemory}$ (Rep(e))
(∀ dicc: $\widehat{dMemory}$)

$$\text{Abs(dicc)} \equiv \text{e}: \widehat{dMemory} \Leftrightarrow \left(\begin{array}{l} \left(\begin{array}{l} titulosDM(dicc) =_{\text{obs}} (clavesDS(e.diccPorTitulo)) \end{array} \right) \wedge_{\text{L}} \\ \left(\begin{array}{l} (\forall n : nombre)definido?DM(dicc, n) =_{\text{obs}} \\ n \in clavesDS(e.diccPorTitulo) \end{array} \right) \wedge_{\text{L}} \\ \left(\begin{array}{l} (\forall n : nombre)(definido?DM(dicc, n) == true) \Rightarrow_{\text{L}} \\ obtenerDM(dicc, n) =_{\text{obs}} \\ obtenerDS(e.diccPorTitulo, n) \end{array} \right) \wedge_{\text{L}} \\ \left(\begin{array}{l} (\forall c : cliente)(\forall p : promesa)(p \in promesasDeDM(dicc, c)) \Leftrightarrow \\ definido?DM(e.diccPorTitulo, titulo(p)) \wedge_{\text{L}} \\ ((tipo(p) == compra \wedge p \in compras(obtenerDM(e.diccPorTitulo, titulo(p))) \vee \\ (tipo(p) == venta \wedge p \in ventas(obtenerDM(e.diccPorTitulo, titulo(p))))) \end{array} \right) \end{array} \right)$$

9.7. Algoritmos

*i*VacioDM()→ $resultado$: $dMemory$
1 $resultado.fueModificado \leftarrow true$
2 $resultado.cliente \leftarrow 0$
3 $resultado.ultimasPromesas \leftarrow vacio()$;
4 $resultado.diccPorTitulo \leftarrow crearDiccString()$;
5 **return** $resultado$
6 **end function**

Complejidad: $\text{O}(1)_{\text{vacio()}} + \text{O}(1)_{\text{crearDiccString}} = \mathbf{O(1)}$

*i*AgregarTituloDM(**inout** e: $estr$, **in** iT: $itConj(titulo)$, **in** num: nat)
1 **var** $infoT$: $infoTitulo$
2 $infoT \leftarrow nuevoInfoTitulo(iT, vacio(), crear(num), \#MaxAcciones(siguiente(iT)))$; $definirDS(e.diccPorTitulo, nomb$
3 **end function**

Complejidad: $\text{O}(1)_{\text{vacio()}} + \text{O}(\text{num})_{\text{crear(n)}} + \text{O}(1)_{\#\text{maxAcciones}} + \text{O}(|\text{nombre(iT)}|)_{\text{definirDS}} =$
O(num + |nombre(iT)|)

```
iDefinido?DM(in e: estr, in nt: nombre)→ resultado: bool
1  return definidoDS?(nt, e.diccPorTitulo)
2  end function
```

Complejidad: $\mathbf{O(|nt|)}_{\text{definido?DS}}$

```
iobtenerDM(in e: estr, in nt: nombre)→ resultado: infoTitulo
1  return obtenerDS(nt, e.diccPorTitulo)
2  end function
```

Complejidad: $\mathbf{O(|nt|)}_{\text{obtenerDS}}$

```
ititulosDM(in e: estr, in nt: nombre)→ resultado: conj(claves)
1  return clavesDS(e.diccPorTitulo)
2  end function
```

Complejidad: $\mathbf{O(\#claves)}_{\text{clavesDS}}$

```
iagregarPromesa(inout dicc: dMemory, in pr: promesa, in cl: cliente, in pN: punteroANat)
 1  e.fueModificado ← true
 2  var datosTitulo: infoTitulo
 3  datosTitulo ← obtenerDS(e.diccPorTitulo, titulo(pr))
 4  var datosPromesa: infoProm
 5  datosPromesa ← nuevoInfoProm(cl, pr, pN)
 6  if tipo(pr) = Venta
 7     then
 8          AgregarRapido(datosPromesa, ventas(datosTitulo))
 9     else
10          insertar(datosPromesa, compras(datosTitulo))
11  end if
12  end function
```

Complejidad:

$O(|\text{titulo(pr)}|)_{\text{obtenerDS}} + O(|\text{titulo(pr)}|)_{\text{nuevoInfoProm}} + O(\text{copy(datosPromesa)})_{\text{agregarRapido}} + O(1)_{\text{insertar}}$
=
Donde O(copy(datosPromesa)) es copiar la tupla de InfoPromesa que es O(copiarPromesa)=O(|titulo(pr)|)
O(|titulo(pr)|)

```
ipromesasDeDM(in e: estr, in cl: cliente)→ resultado: conj(promesa)
 1  if ¬e.fueModificado && e.cliente = c
 2     then
 3          return e.ultimasPromesas
 4     else
 5          var cs: conj(titulo) ← titulosDM(e);
 6          resultado ← vacio();
 7          It ← crearIt(cs)
 8          while (hayProx(It))
 9               var Iventas: ItConj(infoPromesa)
10               Iventas ← crearIt(ventas(obtenerDS(e.diccPorTitulo, titulo(siguiente(It)))))
11               var actual: ItConj(nombre)
12               while (hayProx(Iventas))
```

```
13                  actual ← siguiente(Iventas)
14                  if (cliente(actual) = cl)
15                    then
16                          AgregarRapido(promesa(actual), resultado)
17                    else
18                          avanzar(Iventas)
19                  end if
20              end while
21              var i: Nat ← 1
22              while (i ≤ tam(compras(e.DiccPorTitulo))))
23                  if (dameIndice(compras(e.DiccPorTitulo), i) ≠ NULL &&
24                    cliente(dameIndice(compras(e.DiccPorTitulo), i)) = c)
25                    then
26                          AgregarRapido(promesa(dameIndice(compras(e.DiccPorTitulo), i)), resultado)
27                    else
28                          i++
29                  end if
30              end while
31          end while
32  end if
33  end function
```

Complejidad:

$O(\#titulos)_{titulosDM}$ + $O(1)_{crearIt}$ + O(#titulos)* {
$O(1)_{crearIt}$ + $O(|max\_nt|)$ $_{obtenerDS}$ + O(#Ventas) * $O(copy(promesa))_{AgregarRapido}$ +
O(#Compras) * [$(O(1)_{dameIndice}$ + $O(1)_{dameIndice})$ + $O(copy(promesa))_{AgregarRapido}$ + $O(1)_{dameIndice}$]
})
= O(#titulos) + O(#titulos) * [O(|max_nt|) + O(#Ventas) * O(copy(promesa)) + O(#Compras) *
O(copy(promesa))]
= O(#titulos) * (O(|max_nt|) + O(copy(promesa)) [O(#Ventas) + O(#Compras)])

Ahora, O(copy(promesa)) = O(|titulo(promesa)|) ≤ |nombre(tituloMax)| , y como #Ventas ≤ C ∧ #Compras ≤ C

Complejidad = O(#Titulos) * O(|max_nt|) * O(C) = **O(T * C * |max_nt|)**

9.8. Servicios usados

Informacion de Titulo

Operacion	Aliasing	Complejidad
nuevoInfoTitulo	No	O(copy(V)+copy(C))
titulo	Si	O(1)
ventas	Si	O(1)
compras	Si	O(1)
disponibles	Si	O(1)

Donde V es un conjunto de infoProm y C un arreglo ordenable de infoProm

Informacion de Promesa

Operacion	Aliasing	Complejidad
nuevoinfoProm	No	O(|nombre(t)|)
cliente	No	O(1)
promesa	No	O(1)
acccionesDelCliente	No	O(1)

Usa el Modulo Conjunto Lineal de la Catedra.

10. Módulo Arreglo Ordenable

10.1. Especificación

TAD ARREGLO ORDENABLE(INFOPROM)

igualdad observacional

$$(\forall d, d' : \text{arreglo\_ordenable(infoProm)}) \left(d =_{\text{obs}} d' \iff \begin{array}{c} (tam(d) =_{\text{obs}} tam(d') \\ \wedge\, ultimo(d) =_{\text{obs}} ultimo(d') \\ \wedge\, (\forall i : \text{nat})\ (\text{def?}(d, i) \Rightarrow_{\text{L}} \\ (\text{dameIndice}(i, d) =_{\text{obs}} \text{dameIndice}(i, d')))) \end{array} \right.$$

géneros arreglo_ordenable(infoProm)

exporta arreglo_ordenable(infoProm), generadores, observadores

usa BOOL, NAT, INFOPROM

observadores básicos

dameIndice	: nat i × arreglo_ordenable(infoProm d)	$\longrightarrow$	infoProm	$def?(d, i)$
tam	: arreglo_ordenable(infoProm)	$\longrightarrow$	nat	
ultimo	: arreglo_ordenable(infoProm)	$\longrightarrow$	nat	

generadores

crear	: nat	$\longrightarrow$	arreglo_ordenable(infoProm)	
insertar	: infoProm x × arreglo_ordenable(infoProm) a	$\longrightarrow$	arreglo_ordenable(infoProm)	$ultimo(d) \leq tam(d)$
borrar	: nat i × arreglo_ordenable(infoProm) a	$\longrightarrow$	arreglo_ordenable(infoProm)	$def?(d, i)$

otras operaciones

def?	: arreglo_ordenable(infoProm) d × nat i	$\longrightarrow$	bool	
ordenar	: arreglo_ordenable(infoProm) d	$\longrightarrow$	arreglo_ordenable(infoProm)	
ordenarAux	: arreglo_ordenable(infoProm) d × nat i	$\longrightarrow$	arreglo_ordenable(infoProm)	
dameMin	: arreglo_ordenable(infoProm) d	$\longrightarrow$	arreglo_ordenable(infoProm)	
dameMinAux	arreglo_ordenable(infoProm) d × nat i	$\longrightarrow$	arreglo_ordenable(infoProm)	
damePromesas	arreglo_ordenable(infoProm) d	$\longrightarrow$	conj(infoProm)	
damePromesasAux	reglo_ordenable(infoProm) d × nat i	$\longrightarrow$	conj(infoProm)	

axiomas $\forall d, e$: dicc(nat, significado), $\forall c, k, n$: nat, $\forall s$: significado

dameIndice(i, insertar(x, d))	$\equiv$	**if** ultimo(d) = i **then** x **else** dameIndice(d, i) **fi**
dameIndice(i, borrar(k, d))	$\equiv$	dameIndice(d, i)
tam(crear(n))	$\equiv$	n

tam(insertar(x, d))	≡ tam(d)
tam(borrar(k, d))	≡ tam(d)
ultimo(crear(n))	≡ 1
ultimo(insertar(x, d))	≡ 1 + ultimo(d)
ultimo(borrar(k, d))	≡ ultimo(d)
def?(d, i)	≡ i <ultimo(d)
ordenar(d)	≡ ordenarAux(d, tam(d))
ordenarAux(d, o)	≡ **if** tam(d) = 0 **then** crear(o) **else** insertar(ordenarAux(borrar(dameMin(d), d), o), dameIndice(d, dameMin(d))) **fi**
dameMin(d)	≡ dameMinAux(d, 1)
dameMinAux(d, i)	≡ **if** i = tam(d) **then** tam(d) **else if** totales(dameIndice(i, d)) >totales(dameIndice(dameMinAux(d, i+1), d)) **then** i **else** dameMinAux(d, i + 1) **fi fi**
damePromesas(d)	≡ damePromesasAux(d, 1)
damePromesasAux(d, i)	≡ **if** i >tam(d) **then** ∅ **else if** def?(d, i) **then** Ag(dameIndice(i, d), damePromesasAux(d, i+1) **else** damePromesasAux(d, i + 1) **fi fi**

Fin TAD

10.2. Servicios Exportados

Operacion	Aliasing	Complejidad
Crear	No	O(n)
DameIndice	Si	O(1)
Ordenar	No	O(n * log n)
Insertar	No	O(1)
Borrar	No	O(1)
Tamaño	No	O(1)
DamePromesas	No	O(1)

n es el largo del arreglo

10.3. Interfaz

Interfaz ARREGLO_ ORDENABLE(INFOPROM
Se explica con: TAD Arreglo Ordenable(infoProm)
Géneros: arreglo_ordenable(infoProm)
Operaciones:

Crear(in *n*:nat) → *resultado*: arreglo_ordenable(infoProm)

Pre $\equiv\{\ true\ \}$
Post $\equiv\{\ resultado = crear(n)\ \}$

DameIndice(in i: nat, in d: arreglo_ ordenable(infoProm)) $\rightarrow$ *resultado*: infoProm
Pre $\equiv\{\ def?(d, i)\ \}$
Post $\equiv\{\ resultado = dameIndice(i, d)\ \}$

Ordenar(inout d: arreglo_ ordenable(infoProm))
Pre $\equiv\{\ d_0 = d\ \}$
Post $\equiv\{\ d = ordenar(d_0)\ \}$

Insertar(in x: infoProm, in d: arreglo_ ordenable(infoProm))
Pre $\equiv\{\ d_0 = d \wedge ultimo(d) \leq tam(d) \wedge x \notin damePromesas(d)\ \}$
Post $\equiv\{\ d = insertar(x, d_0)\ \}$

Borrar(in k: nat, in d: arreglo_ ordenable(infoProm))
Pre $\equiv\{\ d_0 = d \wedge def?(d, i)\ \}$
Post $\equiv\{\ d = borrar(k, d_0)\ \}$

Tamaño(in d: arreglo_ ordenable(infoProm)) $\rightarrow$ *resultado*: nat
Pre $\equiv\{\ true\ \}$
Post $\equiv\{\ resultado = tam(d)\ \}$

DamePromesas(in d: arreglo_ ordenable(infoProm)) $\rightarrow$ *resultado*: conj(infoProm)
Pre $\equiv\{\ true\ \}$
Post $\equiv\{\ resultado = damePromesas(d)\ \}$

10.4. Estructura de Representación

arreglo_ordenable(infoProm) se representa con tupla<datos: conj(infoProm), ult: nat, arr: arreglo_dimensionable(itConj(i

10.5. Invariante de Representación

Rep informal:
1 - Todos los naturales mayores que e.ult son NULL
2 - El cardinal de e.datos es menor o igual a e.ult

Rep: $\widehat{arreglo\_ordenable}(infoProm) \rightarrow$ bool
($\forall$ e: $\widehat{arreglo\_ordenable}(infoProm)$) Rep(e) = $true \Leftrightarrow$
e.ult $\geq$ #e.datos $\wedge$ ($\forall$ i: nat)(($i >$e.ult $\Rightarrow_{\text{L}} \neg$ e.arr[i] = NULL))

10.6. Función de Abstracción

Abs: $\widehat{arrego\_ordenable}(infoProm)$ e $\rightarrow$ $\widehat{arreglo\_ordenable}(infoProm)$ (Rep(e))

($\forall$ arr: $\widehat{arrego\_ordenable}(infoProm)$)

$$\text{Abs(arr)} \equiv \text{d}: \widehat{arreglo\_ordenable}(infoProm) \Leftrightarrow \left(\begin{array}{l} \left(\ e.ult =_{obs} ult(d) \wedge tam(d) =_{obs} tam(e.arr) \ \right) \wedge \\ \left(\begin{array}{l} (\forall i : nat) def?(d,i) \Rightarrow_{\text{L}} siguiente(e.arr[i]) = dameIndice(i,d) \wedge \\ \left(\ siguiente(e.arr[i]) \in e.datos \ \right) \end{array} \right) \end{array} \right)$$

10.7. Algoritmos

*i*CREAR(**in** n: nat)$\rightarrow$ $arreglo\_ordenable(infoProm)$

```
1  var arr: arrego_dimensionable(itConj(infoProm));
2  arr ← crearArreglo(n);
3  for (i ← 1 to e.tam)
4        arr[i] ← NULL;
5  end for
6  return ⟨vacio(), 0, crearArreglo(n)⟩;
7  end function
```

Complejidad: $\text{O(n)}_2 + \text{O(n)}_3 * \text{O(1)}_4 = \text{O(n)}$

*i*DAMEINDICE(**in** n: nat, **in** e: $arreglo\_ordenable(infoProm)$)$\rightarrow$ $infoProm$

```
1  return siguiente(e.arr[i]);
2  end function
```

Complejidad: O(1)_1

*i*ORDENAR(**inout** d: $arreglo\_ordenable(infoProm)$)

```
1  return MergeSort(d.arr);
2  end function
```

Complejidad: $\text{O(n * log n)}_{\text{por Merge Sort}}$

*i*MERGESORT(**in/out** A: $arreglo\_dimensionable(itConj(infoProm))$)

```
 1  if (tam(A) > 1)
 2     then
 3          natm ← tam(A)/2;
 4          arreglo_dimensionable(itConj(infoProm))B← copiar(subarreglo(A, 1, m))
 5          arreglo_dimensionable(itConj(infoProm))C← copiar(subarreglo(A, m + 1, tam(A)));
 6          MergeSort(B);
 7          MergeSort(C);
 8          Merge(A, B, C);
 9
10  end if
11  end function
```

iMERGE(**out** A: $arr\_dimen(itConj(iP))$, **in** B: $arr\_dimen(itConj(iP))$, **in** C: $arr\_dimen(itConj(iP))$)

```
1   nat i_b ← 1, i_c ← 1
2   A ← CrearArreglo(tam(B) + tam(C))
3   for (i ← 1 to tam(A))
4         if (i_b ≤ tam(B) ∧ (i_c > tam(C) ∨ (C[i_c] = NULL ∨_L (B[i_b] ≠ NULL ∧_L B[i_b] < C[i_c]))))
5            then A[i] ← B[i_b], i_b ← i_b + 1
6            else A[i] ← C[i_c], i_c ← i_c + 1
7         end if
8   end for
9
10  end function
```

OBS: iP es infoProm y arr_dimen es arreglo_dimensiobable

iINSERTAR(**inout** e: $arreglo\_ordenable(infoProm)$, **in** k: $infoProm$)

```
1   var it: itConj(infoProm);
2   it ← AgregarRapido(e.datos, k);
3   e.arr[e.ult] ← it;
4   e.ult + +;
5   end function
```

Complejidad: $O(1)_1 + O(1)_2 + O(1)_3 + O(1)_4 = O(1)$

iBORRAR(**inout** e: $arreglo\_ordenable(infoProm)$, **in** i: nat)

```
1   EliminarSiguiente(e.arr[i]);
2   e.arr[i] ← NULL;
3   end function
```

Complejidad: $O(1)_1 + O(1)_2 + O(1)_3 = O(1)$

iTAMAÑO(**in** d: $arreglo\_ordenable(infoProm)$) → $resultado$: nat

```
1   return tam(e.arr);
2   end function
```

Complejidad: O(1)

iDAMEPROMESAS(**in** d: $arreglo\_ordenable(infoProm)$) → $resultado$: $conj(infoProm)$

```
1   return e.datos;
2   end function
```

Complejidad: O(1)

10.8. Servicios usados

Utiliza el Modulo Arreglo Dimensionable de la Catedra.