

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

1. Renombres de Módulos	4
2. Módulo Wolfie	4
2.1. Interfaz	4
2.1.1. Parámetros formales	4
2.1.2. Operaciones básicas de wolfie	4
2.2. Representación	5
2.2.1. Representación de wolfie	5
2.2.2. Invariante de representación	5
2.2.3. Función de abstracción	6
2.3. Algoritmos	7
2.3.1. Funciones auxiliares	8
3. Módulo DiccionarioTrie(alpha)	9
3.1. Interfaz	9
3.1.1. Parámetros formales	9
3.1.2. Operaciones básicas de Diccionario String(α)	9
3.1.3. Operaciones básicas del iterador de claves de Diccionario String(α)	9
3.2. Representación	10
3.2.1. Representación del Diccionario String(α)	10
3.2.2. Operaciones auxiliares del invariante de Representación	10
3.2.3. Representación del iterador de Claves del Diccionario String(α)	11
3.3. Algoritmos	11
3.3.1. Algoritmos de Diccionario String	11
3.3.2. Algoritmos del iterador de claves del Diccionario String	12
3.4. Servicios Usados	13
4. Módulo Conjunto Estático de Nats	14
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. Representación de itConjEstNat	15
4.3. TAD CONJUNTO ESTÁTICO DE NATS	16
5. Módulo Promesa	17
5.1. Interfaz	17
5.1.1. Parámetros formales	17
5.1.2. Operaciones básicas de promesa	17
5.2. Representación	17

5.2.1. Representación de promesa	17
5.3. Algoritmos	18
5.3.1. Algoritmos de promesa	18

1. Renombres de Módulos

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

2. Módulo Wolfie

2.1. Interfaz

2.1.1. Parámetros formales

géneros wolfie

se explica con: WOLFIE.

2.1.2. Operaciones básicas de wolfie

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

TÍTULOS(in w : wolfie) $\rightarrow res$: itUni(título)
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{crearItUni}(\text{títulos}(w))\}$
Complejidad: $\Theta(1)$
Descripcion: Devuelve un iterador a los títulos de un wolfie.

PROMESASDE(in c : cliente, in w : wolfie) $\rightarrow res$: itConj(promesa)
Pre $\equiv \{c \in \text{clientes}(w)\}$
Post $\equiv \{res =_{\text{obs}} \text{crearItUni}(\text{promesasDe}(c, w))\}$
Complejidad: $\Theta(T \cdot C \cdot |max_nt|)$
Descripcion: Devuelve un iterador a las promesas de un wolfie

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

INAUGURARWOLFIE(in cs : conj(cliente)) $\rightarrow res$: wolfie
Pre $\equiv \{\neg ?(cs)\}$
Post $\equiv \{res =_{\text{obs}} \text{inaugurarWolfie}(cs)\}$
Complejidad: $\Theta(\#(cs)^2)$
Descripcion: Crea un nuevo wolfie a partir de un conjunto de clientes.

AGREGARTÍTULO(in t : título, in/out w : wolfie)
Pre $\equiv \{w_0 =_{\text{obs}} w \wedge (\forall t2:\text{título}) (t2 \in \text{títulos}(w) \Rightarrow \text{nombre}(t) \neq \text{nombre}(t2))\}$
Post $\equiv \{w =_{\text{obs}} \text{agregarTítulo}(t, w_0)\}$
Complejidad: $\Theta(|\text{nombre}(t)| + C)$

ACTUALIZARCOTIZACIÓN(in nt : 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: $\Theta(C \cdot |nt| + C \cdot \log(C))$

Descripción: Cambia la cotización de un determinado título. Esta operación genera que se desencadene el cumplimiento de promesas (según corresponda): primero de venta y luego, de compra, según el orden descendente de cantidad de acciones por título de cada cliente.

AGREGARPROMESA(**in** c : cliente, **in** p : promesa, **in/out** w : wolfie)

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

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

Complejidad: $\Theta(|\text{título}(p)| + \log(C))$

Descripción: Agrega una nueva promesa.

ENALZA(**in** nt : nombreTítulo, **in** w : wolfie) $\rightarrow res$: bool

Pre $\equiv \{(\exists t: \text{título}) (t \in \text{títulos}(w) \wedge \text{nombre}(t) = nt)\}$

Post $\equiv \{res =_{\text{obs}} \text{enAlza}(nt, w)\}$

Complejidad: $\Theta(|nt|)$

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

2.2. Representación

2.2.1. Representación de wolfie

wolfie se representa con *estr*

donde *estr* es *tupla*(*títulos*: diccTrie(nombre, infoTítulo),
clientes: conjEstNat(cliente)
últimoLlamado: <cliente: cliente, promesas:conj(promesa), fueÚltimo: bool>))

donde *infoTítulo* es *tupla*(*arrayClientes*: array_dimensionable(*tuplaPorCliente*), *cot*: nat, *enAlza*: bool, *maxAcc*: nat, *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 $a < b \Leftrightarrow a.\text{cantAcc} < b.\text{cantAcc}$

2.2.2. Invariante de representación

- (I) Los clientes de *clientes* son los mismos que hay dentro de *títulos*.
- (II) Las promesas de compra son de su título y cliente y no cumplen los requisitos para ejecutarse.
- (III) Las promesas de y venta son de su título y cliente y no cumplen los requisitos para ejecutarse.
- (IV) Las acciones disponibles de cada título son el máximo de acciones de ese título menos la suma de las acciones de ese título que tengan los clientes, y son mayores o iguales a 0.
- (V) El *cliente* de *últimoLlamado* pertenece a *clientes*.
- (VI) En *últimoLlamado*, si *fueÚltimo* es true, las promesas de *promesas* son todas las promesas que tiene *cliente*.
- (VII) Los clientes están ordenados en *arrayClientes* de *e.títulos*.

Rep : *estr* \rightarrow bool

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

2.2.3. Función de abstracción

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

```

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

```

2.3. Algoritmos

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

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

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

```

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

```

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

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

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

```

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

```

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

```

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

```

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

```

1 infoTitulo s  $\leftarrow$  Obtener (nt , e . titulos )
2 if s . cot < cot then
3   s . enAlza  $\leftarrow$  true
4 else
5   s . enAlza  $\leftarrow$  false
6 fi
7 s . cot  $\leftarrow$  cot
8 nat i  $\leftarrow$  0
9 while i < |s . arrayClientes |
10  if (s . arrayClientes [ i ] . promVenta  $\neq$  NULL \yluego limite (*(s . arrayClientes [ i ] . promVenta))
11> s . cot) then
12  s . arrayClientes [ i ] . cantAcc  $\leftarrow$  cantidad (*(s . arrayClientes [ i ] . promVenta))
13  s . accDisponibles  $\leftarrow$  cantidad (*(s . arrayClientes [ i ] . promVenta))
14  s . arrayClientes [ i ] . promVenta = NULL
15  fi
16 endWhile

```

```

17 arreglo_dimensionable(tuplaPorCantAcc)[s.arrayClientes] arr
18 CambiarPorCantAcc(s.arrayClientes, arr)
19 heapsort(arr)
20 i ← 0
21 while i < |s.arrayClientes|
22   if (arr[i].promCompra ≠ NULL \yluego limite(*(arr[i].promCompra)) < s.cot ∧
23 cantidad(*(arr[i].promCompra)) ≤ s.accDisponibles) then
24     arr[i].cantAcc += cantidad(*(arr[i].promCompra))
25     s.accDisponibles -= cantidad(*(arr[i].promCompra))
26     arr[i].promCompra = NULL
27   fi
28   i++
29 endWhile
30 CambiarPorCliente(arr, s.arrayClientes)
31 heapsort(s.arrayClientes)

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

iEnAlza(in nt: nombreT, in e: estr) → res: bool
1  return(Obtener(nt, e.titulos).enAlza)

```

2.3.1. Funciones auxiliares

CrearArrayClientes(in it: itConjEstNat, in n: nat) → res: arreglo_dimensionable(tuplaPorClientes)

```

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

```

CambiarPorCantAcc(in a1: arreglo_dimensionable(tuplaPorCliente), in/out a2: arreglo_dimensionable(tuplaPorCantAcc))

```

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

```

CambiarPorCliente(in a1: arreglo_dimensionable(tuplaPorCantAcc), in/out a2: arreglo_dimensionable(tuplaPorCliente))

```

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

```


3. Módulo DiccionarioTrie(alpha)

3.1. Interfaz

3.1.1. Parámetros formales

géneros string, α

se explica con: DICCIONARIO(String, α), ITERADOR UNIDIRECCIONAL.

géneros: diccString(α), itDicc(diccString).

3.1.2. Operaciones básicas de Diccionario String(α)

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(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(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(longitud(c))

Descripcion: Devuelve el significado con clave c .

Aliasing: No se devuelve una copia del α en res , se devuelve una referencia a la original.

3.1.3. Operaciones básicas del iterador de claves de Diccionario String(α)

CREARIT(in $d : \text{diccString}(\alpha)$) $\rightarrow res : \text{itClaves}(\text{string})$

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

Post $\equiv \{res =_{\text{obs}} \text{crearIt}(d.\text{claves})\}$

Complejidad: O(1)

Descripcion: Crea y devuelve un iterador de claves Diccionario String.

HAYMAS?(in $d : \text{itClaves}(\text{string})$) $\rightarrow res : \text{bool}$

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

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

Complejidad: O(longitud(c))

Descripcion: Informa si hay más elementos por iterar.

ACTUAL(in d : itClaves(string)) $\rightarrow res$: string
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{actual}(it)\}$
Complejidad: $O(\text{longitud}(c))$
Descripcion: Devuelve la clave de la posición actual.

AVANZAR(in/out it : itClaves(string)) $\rightarrow res$: itClaves(α)
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. Representacion

3.2.1. Representación del Diccionario String(α)

diccString(α) se representa con estrDic

donde **estrDic** es tupla(*raiz*: puntero(nodo) *claves*: lista_enlazada(string))

Nodo se representa con estrNodo

donde **estrNodo** es tupla(*valor*: puntero(α) *hijos*: arreglo_estatico[256](puntero(nodo)))

- (I) Existe un único camino entre cada nodo y el nodo raiz (es decir, no hay ciclos).
- (II) Todos los nodos hojas, es decir, todos los que tienen su arreglo hijos con todas sus posiciones en NULL, tienen que tener un valor distinto de NULL.
- (III) Raiz es distinto de NULL
- (IV) En claves está el camino que se recorre desde la raiz hasta cada nodo hoja.

Rep : estrDic \rightarrow bool

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

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

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

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

3.2.2. Operaciones auxiliares del invatriante de Representación

noHayCiclos : puntero(nodo) \rightarrow bool

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

leer : puntero(nodo) \times string \rightarrow bool

```

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

todosNull : arreglo(puntero(nodo))  $\rightarrow$  bool
todosNull( $a$ )  $\equiv$  auxTodosNull( $a, 0$ )
auxTodosNull : arreglo(puntero(nodo))  $\times$  nat  $\rightarrow$  bool
auxTodosNull( $a, i$ )  $\equiv$  if  $i < |a|$  then  $a[i] == \text{NULL} \wedge \text{auxTodosNull}(a, i + 1)$  else  $a[i].\text{valor} == \text{NULL}$  fi

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

todasLasHojas : puntero(nodo)  $\times$  nat  $\rightarrow$  conj(nodo)
todasLasHojas( $p, n$ )  $\equiv$  if  $p == \text{NULL}$  then
    false
else
    if esHoja( $p$ ) then Ag( $*p$ , vacio) else auxTodasLasHojas( $(*p).\text{hijos}, 256$ ) fi
fi

auxTodasLasHojas : arreglo(puntero(nodo))  $\times$  nat  $\rightarrow$  conj(nodo)
auxTodasLasHojas( $a, n$ )  $\equiv$  hojasDeHijos( $a, n, 0$ )
hojasDeHijos : arreglo(puntero(nodo))  $\times$  nat  $\times$  nat  $\rightarrow$  conj(nodo)
hojasDeHijos( $a, n, i$ )  $\equiv$  if  $i = n$  then  $\emptyset$  else todasLasHojas( $a[i]$ )  $\cup$  hojasDeHijos( $a, n, (i + 1)$ ) fi

todasLasHojasTienenValor : puntero(nodo)  $\rightarrow$  bool
todasLasHojasTienenValor( $p$ )  $\equiv$  auxTodasLasHojasTienenValor(todasLasHojas( $p, 256$ ))
auxTodasLasHojasTienenValor : arreglo(puntero(nodo))  $\rightarrow$  bool
auxTodasLasHojasTienenValor( $a$ )  $\equiv$  if  $|a| = 0$  then
    true
else
    dameUno( $a$ ).valor  $\neq \text{NULL} \wedge \text{auxTodasLasHojasTienenValor}(\text{sinUno}(a))$ 
fi

```

3.2.3. Representación del iterador de Claves del Diccionario String(α)

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

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

3.3. Algoritmos

3.3.1. Algoritmos de Diccionario String

```

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

```

Complejidad

```

ICREARNODO()  $\rightarrow$  res = nodo
1  d : arreglo\_estatico[256]
2  i  $\leftarrow$  0

```

```

3  while (i < 256)
4      d[i] ← NULL
5  endwhile
6  hijos ← d
7  valor ← NULL

```

Complejidad

```

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

```

Complejidad

```

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

```

Complejidad

```

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++
7      else
8          return false
9      endif
10 endwhile
11 return p.valor != NULL

```

Complejidad

```

iCLAVES(in diccString( $\alpha$ ): d) → res = lista_enlazada(string)
1 return itClaves(d)

```

Complejidad

3.3.2. Algoritmos del iterador de claves del Diccionario String

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

3.4. Servicios Usados

Módulo	Operación	Complejidad Requerida
arreglo_estático	AgregarElemento	$O(1)$
arreglo_estático	$\bullet[\bullet]$	$O(1)$
lista_enlazada	AgregarAdelante	$O(\text{copy}(\alpha))$
lista_enlazada	$\bullet[\bullet]$	$O(1)$

HEAPSORT(**in/out** arreglo(tuplas): a, **in** int: n)

```

1 fin ← (n-1)
2 while (end > 0)
3     swap(a[fin], a[0])
4     fin ← (fin - 1)
5     bajar(a, 0, fin)
6 endWhile

```

Complejidad

HEAPIFICAR(**in/out** arreglo(tuplas): a, **in** int: n)

```

1 comienzo ← (parteEntera((n-2)/2))
2 while (comienzo > 0)
3     bajar(a, comienzo, n-1)
4     comienzo ← comienzo - 1
5 endWhile

```

Complejidad

BAJAR(**in/out** arreglo(tuplas): a, **in** int: comienzo, **in** int: fin)

```

1 int: raiz
2 int: hijo
3 int: pasaMano
4 raiz ← comienzo
5 while ((raiz * 2) + 1 ≤ fin)
6     hijo ← (raiz * 2) + 1
7     pasaMano ← raiz
8     if (a[pasaMano] < a[hijo])
9         pasaMano ← hijo
10    endIf
11    if ((hijo + 1 ≤ fin) && (a[pasaMano] < a[hijo + 1]))
12        pasaMano ← hijo + 1
13    endIf
14    if (pasaMano != raiz)
15        swap(a[raiz], a[pasaMano])
16        raiz ← pasaMano
17    endIf
18 endWhile

```

Complejidad

BUSQUEDABINARIA(**in** arreglo(tuplas): a, **in** nat: cliente, **in** nat: tam) → res = int

```

1 int: arriba ← tam-1
2 int: abajo ← 0
3 int: centro
4 while (abajo ≤ arriba)
5     centro ← (arriba + abajo)/2;
6     if (arreglo[centro].Π1 == cliente)
7         return 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

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: $\Theta(n * (\log(n)))$

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: $\Theta(n)$

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: $\Theta(n)$

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: $\Theta(1)$

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: $\Theta(1)$

Descripcion: Devuelve la posicion actual

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

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

4.2. Representación

4.2.1. Representación de conjEstNat

conjEstNat se representa con array: **arreglo_dimensionable**(**nat**)

Rep: los elementos estan ordenados y no hay repeticiones

Rep : **array** \rightarrow **bool**

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

Abs : **array** a \rightarrow **conjEstNat**

{**Rep**(a)}

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

estáEnArray? : **nat** \times **arreglo_dimensionable**(**nat**) \times **nat** \rightarrow **bool**

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

4.2.2. Representación de itConjEstNat

itConjEstNat se representa con iterador

donde **iterador** es **tupla**(pos : **nat**, $lista$: **puntero**(**arreglo_dimensionable**(**nat**)))

Rep : **iterador** \rightarrow **bool**

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

Abs : **iterador** it \rightarrow **itConjEstNat**

{**Rep**(it)}

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

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

```

1 itConj(nat)  $it \leftarrow \text{crearIt}(c)$ 
2 arreglo_dimensionable(nat)[cardinal( $c$ )]  $a$ 
3 nat  $i \leftarrow 0$ 
4 while (HaySiguiente?( $it$ ))
5      $a[i] \leftarrow \text{Siguiente}(it)$ 

```

```

6         i++
7         Avanzar(it)
8     endWhile
9     heapsort(a)
10    return(a)

```

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

```

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

```

iCrearIt(in a: array) → res: iterador

```

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

```

iActual(in it: iterador) → res: nat

```

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

```

iActual(in/out it: iterador)

```

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

```

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

```

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

```

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

4.3. 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

$\bullet \in \bullet$: nat × conjEstNat → bool

generadores

crearConjEstNat: conj(nat) → conj(EstNat)

otras operaciones

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

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

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

CREARPROMESA(*in t: nombre, in tipo: tipoPromesa, in n: dinero, in m: nat*) $\rightarrow res : \text{estr}$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{crearPromesa}(t, \text{tipo}, n, m)\}$
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* $\rightarrow \text{bool}$

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

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

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

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

5.3. Algoritmos

5.3.1. Algoritmos de promesa

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) \rightarrow res = estr

1 res.titulo = t

2 res.tipo = tipo

3 res.limite = n

4 res.cantidad = m