

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. Módulo Wolfie	3
1.1. Interfaz	3
1.1.1. Parámetros formales	3
1.1.2. Operaciones básicas de wolfie	3
1.2. Representación	4
1.2.1. Representación de wolfie	4
2. Módulo DiccionarioTrie(alpha)	6
2.1. Interfaz	6
2.1.1. Parámetros formales	6
2.1.2. Operaciones básicas de DiccTrie(α)	6
2.2. Representación	7
2.2.1. Representación del DiccionarioTrie(α)	7
2.3. Algoritmos	7
3. Módulo Conjunto Estático de Nats	7
3.1. Interfaz	7
3.1.1. Operaciones básicas de conjEstNat	8
3.1.2. Operaciones básicas de itConjEstNat	8
3.2. Representación	9
3.2.1. Representación de conjEstNat	9
3.2.2. Representación de itConjEstNat	9

1. Módulo Wolfie

1.1. Interfaz

1.1.1. Parámetros formales

géneros wolfie

se explica con: WOLFIE.

1.1.2. Operaciones básicas de wolfie

CLIENTES(**in** w : wolfie) $\rightarrow res$: itUni(cliente)
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{crearIt}(\text{clientes}(w))\}$
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) $\rightarrow res$: itPromesa(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 : nombreTítulo, **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 $\equiv \{\neg \emptyset?(cs)\}$
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) $\rightarrow res$: 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 : nombreTítulo, **in** cot : nat, **in/out** w : wolfie) $\rightarrow res$: 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: $\Theta(C \cdot |nt| + C \cdot \log(C))$
Descripcion: Cambia la cotización de un determinado título. Esta operación genera que se desencadene el cumplimiento de promesas (según corresponda): primero de venta y luego, de compra, según el orden descendente de cantidad de acciones por título de cada cliente.

AGREGARPROMESA(**in** c : cliente, **in** p : promesa, **in/out** w : wolfie) $\rightarrow res$: wolfie
Pre $\equiv \{w_0 =_{\text{obs}} w \wedge (\exists t:\text{título}) (t \in \text{títulos}(w) \wedge \text{nombre}(t) = \text{título}(p)) \wedge c \in \text{clientes}(w) \wedge_L (\forall p2:\text{promesa}) (p2 \in \text{promesasDe}(c, w) \Rightarrow (\text{título}(p) \neq \text{título}(p2) \vee \text{tipo}(p) \neq \text{tipo}(p2))) \wedge (\text{tipo}(p) = \text{vender} \Rightarrow \text{accionesPorCliente}(c, \text{título}(p),$

$w) \geq \text{cantidad}(p)))\}$

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

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

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.

1.2. Representación

1.2.1. Representación de wolfie

wolfie se representa con estr

donde estr es $\text{tupla}(\text{títulos: diccTrie}(\text{nombre}, \langle \text{arrayClientes: array_dimensionable}(\text{tuplaPorCliente}),$
 $\text{cot: nat},$
 $\text{enAlza: bool},$
 $\text{maxAcc: nat},$
 $\text{accDisponibles: nat}\rangle),$

$\text{clientes: conjEstNat}(\text{cliente})$

$\text{últimoLlamado: } \langle \text{cliente: cliente}, \text{promesas: conj}(\text{promesa}), \text{fueÚltimo: bool}\rangle)$

donde tuplaPorCliente es $\text{tupla}(\text{cliente: cliente}, \text{cantAcc: nat}, \text{promCompra: *promesa}, \text{promVenta: *promesa})$

Con un orden definido por $a < b \Leftrightarrow a.\text{cliente} < b.\text{cliente}$

donde tuplaPorCantAcc es $\text{tupla}(\text{cliente: cliente}, \text{cantAcc: nat}, \text{promCompra: *promesa}, \text{promVenta: *promesa})$

Con un orden definido por $a < b \Leftrightarrow a.\text{cantAcc} < b.\text{cantAcc}$

- (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) $(\forall nt: \text{nombreT}) \left(\text{def?}(nt, e.\text{titulos}) \Rightarrow_{\text{L}} (\text{obtener}(nt, e.\text{titulos}).\text{accDisponibles} = \text{obtener}(nt, e.\text{titulos}).\text{maxAcc} - \text{sumaAccClientes}(\text{obtener}(nt, e.\text{titulos}).\text{arrayClientes}) \wedge \text{obtener}(nt, e.\text{titulos}).\text{accDisponibles} \geq 0) \right) \wedge$
 (V) $(\text{pertenece?}(e.\text{últimoLlamado}.cliente, e.\text{clientes})) \wedge_{\text{L}}$
 (VI) $(e.\text{últimoLlamado}.fueÚltimo \Rightarrow (\forall p: \text{promesa}) \left(\text{pertenece?}(p, e.\text{últimoLlamado}.promesas) \iff (\text{def?}(\text{título}(p), e.\text{titulos}) \wedge_{\text{L}} \right.$
 if $\text{tipo}(p)=\text{compra}$ **then**
 $\text{buscarCliente}(e.\text{últimoLlamado}.cliente, \text{obtener}(\text{título}(p), e.\text{titulos}).\text{arrayClientes}).\text{promCompra} = p$
 else
 $\text{buscarCliente}(e.\text{últimoLlamado}.cliente, \text{obtener}(\text{título}(p), e.\text{titulos}).\text{arrayClientes}).\text{promVenta} = p$
 fi $\left. \right) \wedge$
 (VII) $(\forall t: \text{título}) \text{def?}(t, e.\text{titulos}) \Rightarrow_{\text{L}} ((\forall i: \text{nat}) i < \text{longitud}(\text{buscar}(t, e.\text{titulos}).\text{arrayClientes})-1 \Rightarrow (\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}[i] < (\text{buscar}(t, e.\text{titulos}).\text{arrayClientes}[i+1]))$
 $\text{estáCliente?} : \text{cliente} \times \text{array_dimensionable}(\text{tuplaPorCliente}) \longrightarrow \text{bool}$
 $\text{estáCliente?}(c, a) \equiv \text{auxEstáCliente}(c, a, 0)$
 $\text{auxEstáCliente} : \text{cliente} \times \text{array_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{bool}$
 $\text{auxEstáCliente}(c, a, i) \equiv \text{if } i=\text{longitud}(a) \text{ then false else } a[i].cliente = c \vee \text{auxEstáCliente}(c, a, i+1) \text{ fi}$
 $\text{buscarCliente} : \text{cliente} \times \text{array_dimensionable}(\text{tuplaPorCliente}) \longrightarrow \text{tuplaPorCliente} \quad \{\text{estáCliente}(c, a)\}$
 $\text{buscarCliente}(c, a) \equiv \text{auxBuscarCliente}(c, a, 0)$
 $\text{auxBuscarCliente} : \text{cliente} \times \text{array_dimensionable}(\text{tuplaPorCliente}) \times \text{nat} \longrightarrow \text{tuplaPorCliente} \quad \{\text{estáCliente}(c, a)\}$
 $\text{auxBuscarCliente}(c, a, i) \equiv \text{if } a[i].cliente = c \text{ then } a[i] \text{ else } \text{auxBuscarCliente}(c, a, i+1) \text{ fi}$
 $\text{Abs} : \text{estr } e \longrightarrow \text{wolfie} \quad \{\text{Rep}(e)\}$
 $\text{Abs}(e) =_{\text{obs}} w: \text{wolfie} \mid \text{clientes}(w)=e.\text{clientes} \wedge \text{títulos}(w)=???????? \wedge$
 $(\forall c: \text{cliente}) \text{promesasDe}(c, w)=\text{damePromesas}(\text{crearIt}(e.\text{titulos}), e, c) \wedge$
 $\text{accionesPorCliente}(c, t, w)=\text{buscarCliente}(\text{obtener}(t, e.\text{titulos}).\text{arrayClientes}).\text{cantAcc}$
 $\text{damePromesas} : \text{itTrie}????? \times \text{estr} \times \text{cliente} \longrightarrow \text{conj}(\text{promesa})$
 $\text{damePromesas}(it, e, c) \equiv \text{if hayMas?}(it) \text{ then}$
 if $\text{buscarCliente}(\text{obtener}(\text{actual}(it))).\text{promCompra} \neq \text{NULL}$ **then**
 $\{\text{buscarCliente}(\text{obtener}(\text{actual}(it))).\text{promCompra} \neq \text{NULL}\} \cup \text{fi}$
 if $\text{buscarCliente}(\text{obtener}(\text{actual}(it))).\text{promVenta} \neq \text{NULL}$ **then**
 $\{\text{buscarCliente}(\text{obtener}(\text{actual}(it))).\text{promVenta} \neq \text{NULL}\} \cup \text{fi}$
 $\text{damePromesas}(\text{avanzar}(it), e, c)$
 else
 vacío
 fi

2. Módulo DiccionarioTrie(alpha)

2.1. Interfaz

2.1.1. Parámetros formales

géneros string, α

se explica con: DICCTRIE(α).

géneros: diccTrie(α).

2.1.2. Operaciones básicas de DiccTrie(α)

CREARDICC($()$) $\rightarrow res : \text{diccTrie}(\alpha)$

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

Post $\equiv \{res =_{\text{obs}} \text{vacío}\}$

Complejidad: $\Theta(1)$

Descripcion: Crea un diccionario vacío.

DEFINIR(**in/out** $d : \text{diccTrie}(\alpha)$, **in** $c : \text{string}$, **in** $s : \text{conj}(\alpha)$)

Pre $\equiv \{d =_{\text{obs}} d_0 \wedge \neg \text{definido?}(d, c)\}$

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

Complejidad: $\Theta(\text{longitud}(c))$

Descripcion: Define la clave c con el significado s en el diccionario d .

DEFINIDO?(**in** $d : \text{diccTrie}(\alpha)$, **in** $c : \text{string}$) $\rightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{def?}(c, d)\}$

Complejidad: $\Theta(\text{longitud}(c))$

Descripcion: Devuelve true si y solo si c está definido como clave en el diccionario.

SIGNIFICADO(**in** $d : \text{diccTrie}(\alpha)$, **in** $c : \text{string}$) $\rightarrow res : \alpha$

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

Post $\equiv \{res =_{\text{obs}} \text{obtener}(c, d)\}$

Complejidad: $\Theta(\text{longitud}(c))$

Descripcion: Devuelve el significado con clave c .

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

TODOSLOSSIGNIFICADOS(**in/out** $d : \text{diccTrie}(\alpha)$) $\rightarrow res : \text{conj}(\alpha)$

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

Post $\equiv \{(\forall a : \alpha) a \in res \rightarrow (\exists c : \text{clave}) c \in \text{claves}(d) \wedge_L a = \text{obtener}(d, c)\}$

Complejidad: $\Theta(|\text{max}_c|)$

Descripcion: Devuelve todos los significados guardados en el diccionario d .

Aliasing: res no es modificable

2.2. Representacion

2.2.1. Representación del DiccionarioTrie(α)

diccTrie(α) se representa con dic

donde dic es tupla(*raiz*: puntero(nodoTrie))

NodoTrie se representa con nodo

donde nodo es tupla(*valor*: puntero(α) *hijos*: arreglo(puntero(nodoTrie)))

2.3. Algoritmos

Algorithm 1 iCrear()

```
n : nodo
n ← crearNodo()
raiz ← *n
```

Algorithm 2 iCrearNodo()

```
d : arreglo_estatico[256]
i ← 0
while i < 256 do
    d[i] ← NULL
end while
this.hijos ← d
this.valor ← NULL
```

Algorithm 3 iDefinir(string: s, alfa: a)

```
i ← 0
p ← this.raiz
while i < (longitud(s)) do
    if p.hijos[ord(s[i])] == NULL then
        n : nodo ← crearNodo()
        p.hijos[ord(s[i])] ← *n
    end if
    p ← p.hijos[ord(s[i])]
    i ++
end while
p.valor ← a
```

3. Módulo Conjunto Estático de Nats

3.1. Interfaz

géneros conjEstNat, itConjEstNat

se explica con: CONJUNTO.

Algorithm 4 iObtener(string: s)

```

i ← 0
p ← this.raiz
while i < (longitud(s)) do
    p ← p.hijos[ord(s[i])]
    i ++
end while
p.valor ← a return p.valor

```

3.1.1. Operaciones básicas de conjEstNat

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

PERTENECE?(**in** *n*: nat, **in** *c*: conjEstNat) → *res* : bool
Pre ≡ {true}
Post ≡ {*res* =_{obs} *n* ∈ *c*}
Complejidad: $\Theta(n)$
Descripcion: Crea un conjunto estático de nats

3.1.2. Operaciones básicas de itConjEstNat

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

ACTUAL(**in** *i*: itConjEstNat) → *res* : nat
Pre ≡ {true}
Post ≡ {*res* =_{obs} actual(*i*)}
Complejidad: $\Theta(1)$
Descripcion: Devuelve la posicion actual

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

HAYPRÓX?(**in** *i*: itConjEstNat) → *res* : bool
Pre ≡ {*i*₀ = *i*}
Post ≡ {*res* =_{obs} hayMas?(*i*)}
Complejidad: $\Theta(1)$
Descripcion: Pregunta si hay mas elementos para iterar

3.2. Representación

3.2.1. Representación de conjEstNat

conjEstNat se representa con array: arreglo_dimensionable(nat)

Rep: los elementos estan ordenados y no hay repeticiones

Rep : array \rightarrow bool

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

Abs : array $a \rightarrow$ conjEstNat

{Rep(a)}

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

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

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

3.2.2. Representación de itConjEstNat

itConjEstNat se representa con iterador

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

Rep : iterador \rightarrow bool

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

Abs : iterador $it \rightarrow$ itConjEstNat

{Rep(it)}

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

Algorithm 5 iNuevoConjEstNat(conj)

```

it : itConj
it ← crearIt(conj)
a : arreglo_dimensionable[cardinal(conj)]
i : nat
i ← 0
while haySiguiente?(it) do
    a[i] ← siguiente(it)
    i ++
    avanzar(it)
end while
heapSort(a)
return(a)

```

Servicios usados: se utilizan solo tipos basicos, incluido el arreglo y punteros.

Algorithm 6 iPertenece?(n, c)

```
i : nat  
i ← 0  
b : bool  
b ← false  
while i < longitud(c) do  
    b ← b ∨ c[i] = n  
end while  
return(b)
```

Algorithm 7 iCrearIt(*c*)

```
return < longitud(c), &c >
```

Algorithm 8 iActual(it)

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

Algorithm 9 iPróximo(it)

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

Algorithm 10 ihayPróx(it)

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