

# Algoritmos y Estructura de Datos II

Primer cuatrimestre 2014

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

## Trabajo Practico 2

### Grupo 10

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

### Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

# Índice

<b>1. Módulo Wolfie</b>	<b>3</b>
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
<b>2. Módulo DiccionarioTrie(alpha)</b>	<b>5</b>
2.1. Interfaz	5
2.1.1. Parámetros formales	5
2.1.2. Operaciones básicas de DiccTrie( $\alpha$ )	5
2.2. Representación	6
2.2.1. Representación del DiccionarioTrie( $\alpha$ )	6
2.3. Algoritmos	6

# 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  $\text{tupla}(\text{títulos: diccTrie}(\text{nombre}, \langle \text{arrayClientes: array\_dimensionable}(\text{tuplaPorCliente}),$   
 $\text{cot: nat},$   
 $\text{enAlza: bool},$   
 $\text{maxAcc: nat},$   
 $\text{accDisponibles: nat}\rangle),$   
 $\text{clientes: array\_dimensionable}(\text{cliente})$   
 $\text{últimoLlamado: } \langle \text{cliente: cliente, promesas: conj(promesa), fueÚltimo: bool}\rangle)$

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

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

- (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*.

Rep : estr  $\rightarrow$  bool

$\text{Rep}(e) \equiv \text{true} \iff$   
 (I)  $(\forall c: \text{cliente}) \left( \text{estáCliente?}(c, e.\text{clientes}) \iff (\exists t: \text{título}) \left( \text{def?}(t, e.\text{titulos}) \wedge_{\text{L}} \text{estáCliente?}(c, \text{obtener}(t, e.\text{titulos}).\text{arrayClientes}) \right) \right) \wedge_{\text{L}}$   
 (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)  $(\text{estáCliente}(e.\text{últimoLlamado}.cliente, e.\text{clientes})) \wedge$   
 (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**  $\left. \right)$

## 2. Módulo DiccionarioTrie(alpha)

### 2.1. Interfaz

#### 2.1.1. Parámetros formales

**géneros**    string,  $\alpha$

**se explica con:** DICCTRIE( $\alpha$ ).

**géneros:** diccTrie( $\alpha$ ).

#### 2.1.2. Operaciones básicas de DiccTrie( $\alpha$ )

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  $\alpha$  en  $res$ , se devuelve una referencia a la original.

**TODOSLOSIGNIFICADOS**(**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( $\alpha$ )

$\text{diccTrie}(\alpha)$  se representa con  $\text{dic}$

donde  $\text{dic}$  es  $\text{tupla}(\text{raiz: puntero}(\text{nodoTrie}))$

$\text{NodoTrie}$  se representa con  $\text{nodo}$

donde  $\text{nodo}$  es  $\text{tupla}(\text{valor: puntero}(\alpha) \text{ hijos: arreglo}(\text{puntero}(\text{nodoTrie})))$

## 2.3. Algoritmos

---

### Algorithm 1 iCrear()

---

$n : \text{nodo}$

$n \leftarrow \text{crearNodo}()$

$\text{raiz} \leftarrow *n$

---



---

### Algorithm 2 iCrearNodo()

---

$d : \text{arreglo\_estatico}[256]$

$i \leftarrow 0$

**while**  $i < 256$  **do**

$d[i] \leftarrow \text{NULL}$

**end while**

$\text{this.hijos} \leftarrow d$

$\text{this.valor} \leftarrow \text{NULL}$

---

---

**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
```

---

---

**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
```

---