

# Algoritmos y Estructuras de Datos II

## Diseño

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

### Trabajo Práctico 2: Diseño

Alias del grupo:  
WOWOANERNCTYDOHJTMEA

Integrante	LU	Correo electrónico
Barragán, Jerónimo	1472/21	barragan.jeronimo123@gmail.com
García Alurralde, Jorge	437/22	jalurralde@dc.uba.ar
Mamani Meza, Carlos Ezequiel	496/16	mamanimezacarlos@gmail.com
Sarkissian, Ralph Ezequiel	1698/21	ralphsarkissian.b@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega	Fermín	A
Segunda entrega		

# Índice

<b>1. Definiciones</b>	<b>3</b>
<b>2. Lollapatuza</b>	<b>4</b>
2.1. Interfaz . . . . .	4
2.2. Representación . . . . .	5
2.3. Algoritmos . . . . .	6
<b>3. Puesto De Comidas</b>	<b>11</b>
3.1. Interfaz . . . . .	11
3.2. Representación . . . . .	12
3.3. Algoritmos . . . . .	13
<b>4. Diccionario Digital</b>	<b>17</b>
4.1. Interfaz . . . . .	17
4.2. Representación . . . . .	17
4.3. Algoritmos . . . . .	18

## 1. Definiciones

puestoId es nat

personaId es nat

ítem es nat

cant es nat

## 2. Lollapatuza

### 2.1. Interfaz

se explica con: LOLLAPATUZA.

géneros: lolla.

#### Operaciones básicas de Lollapatuza

CREARLOLLA(in ps: diccLog(puestoId, puesto), in as: conjLineal(personaId)) → res : lolla

Pre ≡ {vendenAlMismoPrecio(ps) ∧ novendieronAun(significados(ps)) ∧ ¬∅?(as) ∧ ¬∅?(claves(ps)) }

Post ≡ {res =<sub>obs</sub> crearLolla(ps, as)}

Complejidad:  $O(P \cdot \log(P) + A \cdot \log(A))$ , donde P es la cantidad de claves de ps, y A es el cardinal de as.

Descripción: Creamos un Lollapatuza a partir de un diccionario válido de puestos y un conjunto valido de personas.

Aliasing: Cualquier modificacion realizada a res afectara los significados de ps. *→ lo podían copiar*

VENDER(in/out l: lolla, in pi: puestoId, in a: personaId, in i: ítem, in c: cant)

Pre ≡ {l =<sub>obs</sub> L<sub>0</sub> ∧ a ∈ personas(l) ∧ def?(pi, puestos(l)) ∧<sub>L</sub> (haySuficiente?(obtener(pi, puestos(l)), i, c) ∧ i ∈ menu(obtener(pi, puestos(l))))}

Post ≡ {l =<sub>obs</sub> vender(L<sub>0</sub>, pi, a, i, c)}

Complejidad:  $O(\log(A) + \log(I) + \log(P) + \log(cant))$  ✓

Descripción: Registra la compra de una cantidad de un ítem particular, realizada por una persona en un puesto.

Aliasing: No genera aliasing.

HACKEAR(in/out l: lolla, in a: personaId, in i: ítem)

Pre ≡ {l =<sub>obs</sub> L<sub>0</sub> ∧ consumoSinPromoEnAlgúnPuesto(l, a, i)} ✓

Post ≡ {l =<sub>obs</sub> hackear(L<sub>0</sub>, a, i)}

Complejidad:  $O(\log(A) + \log(I))$ . En caso de que el puesto correspondiente deje de ser hackeable para esa persona e ítem luego de esta operación, la complejidad sera  $O(\log(A) + \log(I) + \log(P))$

Descripción: Realiza acción de hackeo de sistema, el cual remueve una unidad del ítem i si la compra fue realizada sin promoción.

Aliasing: No genera aliasing.

GASTOPERSONA(in l: lolla, in a: personaId) → res : dinero

Pre ≡ {a ∈ personas(l)}

Post ≡ {res =<sub>obs</sub> gastoTotal(l, a)}

Complejidad:  $O(\log(A))$  ✓

Descripción: Devuelve el gasto total de la persona.

Aliasing: No genera aliasing.

MASGASTÓ(in l: lolla) → res : personaId

Pre ≡ {¬Vacio(l.personas)}

Post ≡ {res =<sub>obs</sub> másGastó(l)}

Complejidad:  $O(1)$  ✓

Descripción: Dada un festival, se devuelve el ID de la persona que mas dinero gastó. Si hay mas de una persona con el mismo gasto, se devuelve la de menor ID.

Aliasing: No genera aliasing.

PERSONAS(in l: lolla) → res : conjLineal(personaId)

Pre ≡ {true}

Post ≡ {res =<sub>obs</sub> personas(l)} ✓

Complejidad:  $O(1)$

Descripción: Devuelve el conjunto de personas del festival.

Aliasing: Se devuelve una referencia no modificable.

PUESTOS(in l: lolla) → res : diccLog(puestoId, puesto) *→ PuestoId?*

Pre ≡ {true}

Post ≡ {res =<sub>obs</sub> puestos(l)}

Complejidad:  $O(1)$

Descripción: Se devuelve un diccionario con los puestos

Aliasing: Se devuelve una referencia no modificable. ✓

**PUESTOCONMENORSTOCKDEÍTEM**(in  $l$ : lolla, in  $i$ : ítem)  $\rightarrow res$  : puestoId

**Pre**  $\equiv \{\neg \text{Vacío}(l.\text{puestos})\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{menorStock}(l, i)\}$

**Complejidad**:  $O(P * \log(I))$

**Descripción**: Devuelve el ID del puesto de menor stock para un ítem dado. Si hay más de un puesto que tiene stock mínimo, devuelve el de menor ID.

**Aliasing**: No genera aliasing.

**PERTENECEPERSONA**(in  $l$ : lolla, in  $a$ : personaId)  $\rightarrow res$  : bool

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

**Post**  $\equiv \{res =_{\text{obs}} (a \in \text{personas}(l))\}$

**Complejidad**:  $O(\log(A))$

**Descripción**: Indica si la persona está en el festival.

**Aliasing**: No genera aliasing.

**PERTENECEPUESTO**(in  $l$ : lolla, in  $pi$ : puestoId)  $\rightarrow res$  : bool

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

**Post**  $\equiv \{res =_{\text{obs}} \text{def?}(\text{puestos}(l), pi)\}$

**Complejidad**:  $O(\log(P))$

**Descripción**: Indica si el puesto está en el festival.

**Aliasing**: No genera aliasing.

**VENDENALMISMOPRECIO**(in  $ps$ : diccLog(puestoId, puesto))  $\rightarrow res$  : bool

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

**Post**  $\equiv \{res =_{\text{obs}} \text{true} \iff \text{vendenAlMismoPrecio}(\text{significados}(ps))\}$

**Complejidad**:  $O(P \times I \times \log(I))$

**Descripción**: Dado un diccionario de puestos, verifico si cada ítem que vende alguno, tiene el mismo precio que en los otros que lo también venden.

**Aliasing**: No genera aliasing.

## 2.2. Representación

### Representación de Lollapatuza

**lolla se representa con estr**

donde **estr** es  $\text{tupla}(\text{itPuestos: diccLog(puestoId, itDicc(puestoId, puesto))},$   
 $\text{puestos: diccLog(puestoId, puesto)},$   
 $\text{conjPersonas: conjLineal(personaId)},$   
 $\text{personas: diccLog(personaId, infoPersona)},$   
 $\text{másGastadores: diccLog(tupla\langle \text{gasto} : \text{int}, \text{personaId} \rangle, \text{personaId})}$ )

donde **infoPersona** es  $\text{tupla}(\text{gastoPersona: nat},$   
 $\text{comprasSinDescuento: diccLog(ítem, diccLog(puestoId, tupla\langle \text{cantidad: nat}, \text{itPuesto} : \text{itDicc(puestoId, puesto)} \rangle))})$ )

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

**Rep**( $e$ )  $\equiv \text{true} \iff 1 \wedge_L 2 \wedge_L 3 \wedge_L 4 \wedge_L 5 \wedge_L 6 \wedge_L 7 \wedge_L 8 \wedge_L 9$

1.  $\neg \emptyset(e.\text{personas})$
2.  $\text{claves}(e.\text{personas}) = e.\text{conjPersonas}$
3.  $\text{claves}(e.\text{puestos}) = \text{claves}(e.\text{itPuestos})$
4.  $(\forall \text{personaId} : \text{nat})(\text{personaId} \in e.\text{conjPersonas} \Rightarrow_L \text{sumaDeGastos}(\text{personaId}, \text{significados}(e.\text{puestos})) = \text{obtener}(\text{personaId}, e.\text{personas}).\text{gastoPersona})$
5.  $(\forall \text{puestoId} : \text{nat})(\text{def?}(\text{puestoId}, e.\text{itPuestos}) \Rightarrow_L (\pi_1(\text{Siguiente}(\text{obtener}(\text{puestoId}, e.\text{itPuestos}))) = \text{puestoId} \wedge \pi_2(\text{Siguiente}(\text{obtener}(\text{puestoId}, e.\text{itPuestos}))) = \text{obtener}(\text{puestoId}, e.\text{puestos})))$
6.  $(\forall t : \text{tupla}(\text{int}, \text{nat}))(\text{def?}(t, e.\text{masGastadores}) \Rightarrow_L (t.\text{personaId} \in e.\text{conjPersonas} \wedge_L \text{obtener}(t.\text{personaId}, e.\text{personas}).\text{gastoPersona} = t.\text{gasto}))$
7.  $\# \text{claves}(e.\text{masGastadores}) = \# e.\text{conjPersona}$

una desc. en lenguaje natural vendría bien

8.  $(\forall a : \text{persona})(\forall i : \text{item})(\forall p : \text{puestoId})$   
 $(a \in e.\text{conjPersonas} \wedge \text{def?}(p, e.\text{puestos}) \wedge_L i \in \text{menu}(\text{obtener}(p, e.\text{puestos})) \Rightarrow_L$   
 $(\# \text{unidadesSinDescuento}(a, i, \text{obtener}(p, e.\text{puestos}), \text{ventas}(\text{obtener}(p, e.\text{puestos}), a)) > 0 \Rightarrow$   
 $(\text{def?}(i, \text{obtener}(a, e.\text{personas}).\text{comprasSinDescuento}) \wedge_L \text{def?}(p, \text{obtener}(i, \text{obtener}(a, e.\text{personas}).\text{comprasSinDescuento}))$   
 $\wedge_L \pi_1(\text{obtener}(p, \text{obtener}(i, \text{obtener}(a, e.\text{personas}).\text{comprasSinDescuento}))) = \# \text{unidadesSinDescuento}(a, i, \text{obte-}$   
 $\text{ner}(p, e.\text{puestos}), \text{ventas}(p, a)) \wedge \text{Siguiente}(\pi_2(\text{obtener}(p, \text{obtener}(i, \text{obtener}(a, e.\text{personas}).\text{comprasSinDescuento}))))$   
 $= \text{obtener}(p, e.\text{itPuestos}))))$

9.  $(\forall a : \text{persona})(\forall i : \text{item})(\forall p : \text{puestoId})$   
 $(a \in e.\text{conjPersonas} \wedge_L \text{def?}(i, \text{obtener}(a, e.\text{personas}).\text{comprasSinDescuento}) \wedge_L \text{def?}(p, \text{obtener}(i, \text{obtener}(a,$   
 $e.\text{personas}).\text{comprasSinDescuento})) \Rightarrow_L (\text{def?}(p, e.\text{puestos}) \wedge_L i \in \text{menu}(\text{obtener}(p, e.\text{puestos})) \wedge$   
 $\text{obtener}(p, \text{obtener}(i, \text{obtener}(a, e.\text{personas}).\text{comprasSinDescuento})) =$   
 $\langle \text{unidadesSinDescuento}(a, i, \text{obtener}(p, e.\text{puestos}), \text{ventas}(\text{obtener}(p, e.\text{puestos}), a)), \text{obtener}(p, e.\text{itPuestos}) \rangle))$

$\text{sumaDeGastos} : \text{persona } p \times \text{multiconj}(\text{puesto}) c \rightarrow \text{nat}$

$\text{sumaDeGastos}(p, c) \equiv \text{if } \emptyset?(c) \text{ then } 0 \text{ else } \text{gastosDe}(\text{dameUno}(c), p) + \text{sumaDeGastos}(p, \text{sinUno}(c)) \text{ fi}$

$\# \text{unidadesSinDescuento} : \text{persona } per \times \text{item } i \times \text{puesto } p \times \text{multiconj}(\langle \text{item} \times \text{cant} \rangle) m \rightarrow \text{nat}$

$\{i \in \text{menu}(p) \wedge m \subset \text{ventas}(p, per)\}$

$\# \text{unidadesSinDescuento}(per, i, p, m) \equiv \text{if } \emptyset?(m) \text{ then } 0$

$\text{else if } \text{consumióSinPromo1Venta?}(p, i, \text{dameUno}(m))$

$\text{then } \pi_2(\text{dameUno}(m)) + \# \text{unidadesSinDescuento}(per, i, p, \text{sinUno}(m))$

$\text{else } \# \text{unidadesSinDescuento}(per, i, p, \text{sinUno}(m)) \text{ fi}$

$\text{fi}$

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

$\text{Abs}(e) \equiv l : \text{lolla} / \text{puestos}(l) = e.\text{puestos} \wedge \text{personas}(l) = e.\text{conjPersonas}$

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

## 2.3. Algoritmos

### Algoritmos del módulo

Definimos una relación de orden entre  $\text{tupla}\langle \text{int}, \text{nat} \rangle$  aprovechando que al crear un iterador a un  $\text{diccLog}$ , la siguiente clave es la mínima. Aclaración: Por cómo se construye la relación de orden, la mínima clave se corresponde con la persona que más gastó.

(en lugar de guardarse negativos, podrían invertir el < en la relación de orden)

$\text{io} < \bullet (\text{in } A : \text{tupla}\langle a_1 : \text{int}, a_2 : \text{nat} \rangle, \text{in } B : \text{tupla}\langle b_1 : \text{int}, b_2 : \text{nat} \rangle) \rightarrow \text{res} : \text{bool}$

1:  $\text{res} \leftarrow a_1 < b_1 \parallel (a_1 = b_1 \ \&\& \ a_2 < b_2)$

$\triangleright O(1)$

Complejidad:  $O(1)$

---

**iCrearLolla**(in  $ps$ : diccLog(puestoId, puesto), in  $as$ : conj(personaId))  $\rightarrow res$ : estr

```

1: itPuestos  $\leftarrow$  VACIO()  $\triangleright O(1)$ 
2: puestos  $\leftarrow$  VACIO()  $\triangleright O(1)$ 
3: it1  $\leftarrow$  crearIt(ps)  $\triangleright O(1)$ 
4: while haySiguiente(it1) do  $\triangleright$  costo ciclo:  $O(P * \log(P))$ 
5:   it2  $\leftarrow$  definir(puestos, siguienteClave(it1), siguienteSignificado(it1))
6:   definir(itPuestos, siguienteClave(it1), it2)
7:   Avanzar(it1)  $\triangleright O(1)$ 
8: end while
9: personas  $\leftarrow$  VACIO()  $\triangleright O(1)$ 
10: masGastadores  $\leftarrow$  VACIO()  $\triangleright O(1)$ 
11: It3  $\leftarrow$  crearIt(as)  $\triangleright O(1)$ 
12: while haySiguiente(it3) do  $\triangleright$  costo ciclo:  $O(A * \log(A))$ 
13:   definir(personas, siguiente(it3), <0, VACIO()>)
14:   definir(masGastadores, <0, siguiente(it3)>, siguiente(it3))
15:   Avanzar(it3)
16: end while
17: res  $\leftarrow$  <itPuestos, puestos, as, personas, masGastadores>

```

Complejidad:  $O(P \times \log(P) + A \times \log(A))$

Justificación: Donde P es la cantidad de claves de ps, y A es el cardinal de as.

(\*) *comprasHackeables no está declarado (o debían cerrar el if de arriba primero)*

---

**iVender**(in/out  $l$ : estr, in  $pi$ : puestoId, in  $a$ : personaId, in  $i$ : ítem, in  $c$ : cant)

```

1: puesto  $\leftarrow$  significado(estr.puestos, pi)  $\triangleright$  Referencia al puesto;  $O(\log(P))$  ✓
2: gastoVenta  $\leftarrow$  Vender(puesto, a, i, cant)  $\triangleright O(\log(A)) + O(\log(I)) + O(\log(cant))$  ✓
3: infoPersona  $\leftarrow$  significado(estr.personas, a)  $\triangleright$  Referencia modificable;  $O(\log(A))$  ✓
4: gastoActual  $\leftarrow$  infoPersona.gastoPersona  $\triangleright O(1)$ 
5: nuevoGasto  $\leftarrow$  infoPersona.gastoPersona + gastoVenta  $\triangleright O(1)$ 
6: BORRAR(estr.masGastadores, <-gastoActual, a>)  $\triangleright O(\log(A))$  ✓
7: DEFINIR(estr.masGastadores, <-nuevoGasto, a>, a)  $\triangleright$  Actualización de másGastadores  $O(\log(A))$  ✓
8: infoPersona.gastoPersona  $\leftarrow$  nuevoGasto  $\triangleright$  Actualizar info persona;  $O(1)$ 
9: descuento  $\leftarrow$  obtenerDescuento(p, i, cant)  $\triangleright O(\log(I)) + O(\log(cant))$  ✓
10: if descuento = 0  $\triangleright$  Modificar comprasSinDescuento
11:   then if  $\neg$ definido?(infoPersona.comprasSinDescuento, i)  $\triangleright O(\log(I))$ 
12:     then definir(infoPersona.comprasSinDescuento, i, Vacío())  $\triangleright O(\log(I))$ 
13:     comprasHackeables  $\leftarrow$  significado(infoPersona.comprasSinDescuento, i)  $\triangleright O(\log(I))$ 
14:   else if  $\neg$ definido?(comprasHackeables, pi)
15:     then itPuesto  $\leftarrow$  obtener(estr.itPuestos, pi)  $\triangleright O(\log(P))$ 
16:     definir(comprasHackeables, pi, <0, itPuesto>)  $\triangleright O(\log(P))$ 
17:   else cantidad  $\leftarrow$  significado(comprasHackeables, pi).cantidad  $\triangleright$  Ref. mod.;  $O(\log(P))$ 
18:   fi
19: fi
20: else cantidad  $\leftarrow$  cantidad + cant  $\triangleright O(1)$ 
21: fi

```

Complejidad:  $O(\log(A) + \log(I) + \log(P) + \log(cant))$

---



---

**iHackear**(in/out  $l$ : estr, in  $a$ : personaId, in  $i$ : ítem)

```

1: infoPersona ← significado(estr.personas, a)                                ▷ Ref modificable;  $O(\log(A))$  ✓
2: puestosHackeables ← significado(infoPersona.comprasSinDescuento, i)      ▷  $O(\log(I))$  ✓
3: itMin ← crearIt(puestosHackeables)                                       ▷ (clave, significado);  $O(1)$  ✓
4: id_puesto ← siguienteClave(itMin)                                         ▷  $O(1)$  ✓
5: itPuesto ← siguienteSignificado(itMin).itPuesto                          ▷ obtengo: it al dicc (clave, significado);  $O(1)$  ✓
6: puesto ← siguienteSignificado(itPuesto) //  $O(1)$ 
7: precioItem ← precio(puesto, i)                                           ▷ Actualizo infoPersona y masGastadores;  $O(\log(I))$  ✓
8: gastoActual ← infoPersona.gastoPersona                                ▷ Tipo primitivo en tupla por referencia se pasa por copia;  $O(1)$ 
9: BORRAR(masGastadores, < -gastoActual, a >)                               ▷  $O(\log(A))$  ✓
10: gastoActual ← infoPersona.gastoPersona - precioItem                    ▷  $O(1)$ 
11: DEFINIR(masGastadores, < -gastoActual, a >, a)                           ▷ Agrego;  $O(\log(A - 1)) = O(\log(A))$  ✓
12: infoPersona.gastoPersona ← gastoActual                                  ▷  $O(1)$ 
13: OlvidarItem(puesto, a, i)                                               ▷  $O(\log(A)) + O(\log(I))$  ✓
14: itemsHackeables ← siguienteSignificado(itMin).cantidad - 1              ▷  $O(1)$ 
15: if itemsHackeables > 0                                                  ▷ ¿Sigue siendo hackeable el puesto?
16:   then siguienteSignificado(itMin) ← < itemsHackeables, itPuesto >      ▷  $O(1)$  ✓
17:   else BORRAR(puestosHackeables, id_puesto)                             ▷  $O(\log(P))$ 
18: fi

```

Complejidad:  $O(\log(A) + \log(I)) \vee O(\log(A) + \log(I) + \log(P))$

Justificación: Antes del if tenemos complejidad  $O(\log(A) + \log(I))$ ; si la guarda es verdadera,  $O(\text{if}) = O(1) \rightarrow O(\log(A) + \log(I)) + O(1) = O(\log(A) + \log(I))$ ; si no,  $O(\text{if}) = O(\log(P)) \rightarrow O(\log(A) + \log(I)) + O(\log(P)) = O(\log(A) + \log(I) + \log(P))$  ✓

---



---



---

**igastoPersona**(in  $l$ : estr, in  $a$ : personaId)  $\rightarrow$  res : nat

```

1: dinero ← Significado(estr.personas, a).gastoPersona                    ▷  $O(\log(A)) + O(1)$ 

```

Complejidad:  $O(\log(A))$  ✓

---



---



---

**imásGastó**(in  $l$ : estr)  $\rightarrow$  res : personaId

```

1: itMin ← crearIt(estr.masGastadores)                                     ▷  $O(1)$ 
2: res ← Copiar(SiguienteSignificado(itMin))                             ▷  $O(1)$ 

```

Complejidad:  $O(1)$  ✓

---



---



---

**ipuestoConMenorStockDeÍtem**(in  $l$ : estr, in  $i$ : ítem)  $\rightarrow$  res : puestoId

```

1:  $it \leftarrow crearIt(estr.puestos)$ 
2:  $id\_puestoMenorStock \leftarrow siguienteClave(it)$   $\triangleright$  puestoId
3:  $puestoMenorStock \leftarrow siguienteSignificado(it)$ 
4:  $menorStock \leftarrow obtenerStock(puestoMenorStock, i)$   $\triangleright O(\log(I))[*]$ 
5:  $avanzar(it)$ 
6: while haySiguiente(it) do  $\triangleright P \text{ veces} \rightarrow P * O(\log(I)) = O(P \times \log(I))$  ✓
7:    $id\_puestoActual \leftarrow siguienteClave(it)$ 
8:    $puestoActual \leftarrow siguienteSignificado(it)$ 
9:   if estaEnMenu?(puestoActual, i)  $\triangleright O(\log(I))$ 
10:    then  $stockActual \leftarrow obtenerStock(puestoActual, i)$   $\triangleright O(\log(I))$ 
11:    if  $stockActual < menorStock$ 
12:      then  $menorStock \leftarrow stockActual$ 
13:       $id\_puestoMenorStock \leftarrow siguienteClave(it)$ 
14:      if  $stockActual = menorStock \ \&\& \ id\_puestoActual < id\_puestoMenorStock$ 
15:        then  $menorStock \leftarrow stockActual$ 
16:         $id\_puestoMenorStock \leftarrow id\_puestoActual$ 
17:     $avanzar(it)$ 
18: end while
19:  $res \leftarrow copiar(id\_puestoMenorStock)$ 

```

Complejidad:  $O(P \times \log(I))$

Justificación:  $[*]$  Para consultar stock,  $i$  debe pertenecer a puesto (pasa lo mismo cuando inicializo menorStock).

---



---

**ipertenecePersona**(in  $l$ : estr, in  $a$ : personaId)  $\rightarrow$  res : bool

```

1:  $res \leftarrow pertenece?(estr.personas, a)$   $\triangleright O(\log(A))$ 

```

Complejidad:  $O(\log(A))$

---



---

**ipertenecePuesto**(in  $l$ : estr, in  $pi$ : puestoId)  $\rightarrow$  res : bool

```

1:  $res \leftarrow definido?(estr.puestos, pi)$   $\triangleright O(\log(P))$ 

```

Complejidad:  $O(\log(P))$

---



---

**iPersonas**(in  $l$ : estr)  $\rightarrow$  res : conjLineal(personaId) no modificable

```

1:  $res \leftarrow estr.conjPersonas$   $\triangleright$  Referencia no modificable;  $O(1)$ 

```

Complejidad:  $O(1)$

---



---

**iPuestos**(in  $l$ : estr)  $\rightarrow$  res : diccLog(puestoId, puesto) no modificable

```

1:  $res \leftarrow estr.puestos$   $\triangleright$  Referencia no modificable;  $O(1)$ 

```

Complejidad:  $O(1)$

(no hacía falta implementarla)

---

**ivendenAlMismoPrecio**(in  $ps : \text{DiccLog}(\text{puestoId}, \text{puesto}) \rightarrow res : \text{bool}$

```
1:  $it \leftarrow \text{crearIt}(ps)$   $\triangleright \Theta(1)$ 
2:  $id\_puestoActual \leftarrow \text{siguienteClave}(it)$ 
3:  $puestoActual \leftarrow \text{siguienteSignificado}(it)$ 
4:  $diccLog : \text{todosLosItems} \leftarrow \text{vacío}()$  ✓
5:  $\text{agregarItemsNoContenidos}(\text{todosLosItems}, \text{puestoActual.menu})$  ✓
6:  $\text{avanzar}(it)$ 
7:  $res \leftarrow \text{true}$ 
8: while  $\text{haySiguiente}(it) \ \&\& \ res$  do  $\triangleright P \text{ veces} \rightarrow P \times 2 \cdot O(I \times \log(I)) = O(P \times I \times \log(I))$ 
9:    $id\_puestoActual \leftarrow \text{siguienteClave}(it)$ 
10:   $puestoActual \leftarrow \text{siguienteSignificado}(it)$ 
11:   $res \leftarrow res \ \&\& \ \text{aMismoItemMismoPrecio}(\text{todosLosItems}, \text{puestoActual.menu})$ 
12:   $\text{agregarItemsNoContenidos}(\text{todosLosItems}, \text{puestoActual.menu})$  ✓
13:   $\text{avanzar}(it)$ 
14: end while
```

Complejidad:  $O(P \times I \times \log(I))$

---

Esta es una operación privada que recibe dos menús, agregando al primer menú los del segundo que no están definidos el primero:

---

**iagregarItemsNoContenidos**(in/out  $granMenu : \text{diccLog}(\text{ÍtemId}, \text{nat})$ , in  $menuMenor : \text{diccLog}(\text{ÍtemId}, \text{nat})$ )

```
1:  $it\_menuMenor \leftarrow \text{crearIt}(menuMenor)$   $\triangleright O(1)$ 
2:  $id\_itemMenuMenor \leftarrow \text{siguienteClave}(it\_menuMenor)$  ✓
3:  $precioMenuMenor \leftarrow \text{siguienteSignificado}(it\_menuMenor)$ 
4: while  $\text{haySiguiente}(it\_menuMenor)$  do  $\triangleright$  Peor caso, recorro todos los ítems del festival  $O(I \times \log(I))$ 
5:   if  $\neg \text{definido?}(granMenu, id\_itemMenuMenor)$ 
6:     then  $\text{definir}(granMenu, id\_itemMenuMenor, precioMenuMenor)$   $\triangleright O(\log(I))$ 
7:   fi ✓
8:    $\text{avanzar}(it\_menuMenor)$ 
9: end while
```

Complejidad:  $O(I \times \log(I))$

---

Esta es una operación privada que recibe dos menús, verificando si todos los ítems del segundo cuestan lo mismo que en el primero: ✓

---

**iaMismoItemMismoPrecio**(in  $granMenu : \text{diccLog}(\text{ÍtemId}, \text{nat})$ , in  $menuMenor : \text{diccLog}(\text{ÍtemId}, \text{nat}) \rightarrow res : \text{bool}$

```
1:  $it\_menuMenor \leftarrow \text{crearIt}(menuMenor)$   $\triangleright O(1)$ 
2:  $id\_itemMenuMenor \leftarrow \text{siguienteClave}(it\_menuMenor)$ 
3:  $precioMenuMenor \leftarrow \text{siguienteSignificado}(it\_menuMenor)$ 
4:  $res \leftarrow \text{true}$ 
5: while  $\text{haySiguiente}(it\_menuMenor) \ \&\& \ res$  do  $\triangleright$  Peor caso, recorro todos los ítems del festival  $O(I \times \log(I))$ 
6:   if  $\text{definido?}(granMenu, id\_itemMenuMenor)$ 
7:     then  $res \leftarrow res \ \&\& \ (\text{significado}(granMenu, id\_itemMenuMenor) = precioMenuMenor)$   $\triangleright O(\log(I))$ 
8:   fi
9:    $\text{avanzar}(it\_menuMenor)$  ✓
10: end while
```

Complejidad:  $O(I \times \log(I))$

---

### 3. Puesto De Comidas

#### 3.1. Interfaz

se explica con: PUESTODECOMIDA.

géneros: puesto.

#### Operaciones básicas de Puesto De Comida

**CREARPUESTO**(in *menu*: diccLog(ítem, nat), in *stock*: diccLog(ítem, nat), in *promos*: diccLog(ítem, diccLog(nat, nat)))  $\rightarrow res$  : puesto

**Pre**  $\equiv \{claves(menu) =_{obs} claves(stock) \wedge claves(promos) \subset claves(menu)\}$

**Post**  $\equiv \{res =_{obs} crearPuesto(menu, stock, promos)\}$

**Complejidad**:  $O(I \times (D \times \log(N) + N \times \log(N)))$ . Donde N es la maxima cantidad para la que esta definida un descuento de un item en promos. D es el maximo cardinal de los descuentos pasados por parametro, definidos para un item.

**Descripción**: Crea nuevo puesto de comida. ✓

**Aliasing**: Devuelve una referencia modificable.

**VENDER**(in/out *p*: puesto, in *a*: idPersona, in *i*: ítem, in *cant*: nat)  $\rightarrow res$  : nat

**Pre**  $\equiv \{p =_{obs} p_0 \wedge haySuficiente?(p_0, i, cant)\}$

**Post**  $\equiv \{p =_{obs} vender(p_0, a, i, cant)\}$

**Complejidad**:  $O(\log(A) + \log(I) + \log(cant))$  ✓

**Descripción**: Realiza la venta de una cantidad determinada del item, para una persona. Se devuelve como resultado el gasto de la venta.

**Aliasing**: No genera aliasing.

**PRECIO**(in *p*: puesto, in *i*: ítem)  $\rightarrow res$  : nat

**Pre**  $\equiv \{i \in \text{menú}(p)\}$

**Post**  $\equiv \{res =_{obs} precio(p, i)\}$

**Complejidad**:  $O(\log(I))$  ✓

**Descripción**: Devuelve el precio del item en el puesto p.

**Aliasing**: No genera aliasing.

**OBTENERSTOCK**(in *p*: puesto, in *i*: ítem)  $\rightarrow res$  : nat

**Pre**  $\equiv \{i \in \text{menú}(p)\}$

**Post**  $\equiv \{res =_{obs} stock(p, i)\}$

**Complejidad**:  $O(\log(I))$  ✓

**Descripción**: Devuelve el stock de un ítem en un puesto determinado.

**Aliasing**: No genera aliasing.

**OBTENERDESCUENTO**(in *p*: puesto, in *i*: ítem, in *c*: cant)  $\rightarrow res$  : nat

**Pre**  $\equiv \{i \in \text{menú}(p)\}$

**Post**  $\equiv \{res =_{obs} descuento(p, i, c)\}$

**Complejidad**:  $O(\log(cant) + \log(I))$  ✓

**Descripción**: Devuelve el descuento a aplicarse sobre la compra de una cierta cantidad de ítems en un puesto. Si no hay un descuento definido para esa cantidad, se devuelve 0.

**Aliasing**: No genera aliasing.

**OBTENERGASTO**(in *p*: puesto, in *a*: personaId)  $\rightarrow res$  : nat

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

**Post**  $\equiv \{res =_{obs} gastosDe(p, a)\}$

**Complejidad**:  $O(\log(A))$  ✓

**Descripción**: Devuelve el menú del puesto de comida.

**Aliasing**: No genera aliasing.

**OLVIDARÍTEM**(in/out *p*: puesto, in *a*: personaId, in *i*: ítem)

**Pre**  $\equiv \{p =_{obs} P_0 \wedge_L i \in \text{menu}(P_0) \wedge_L \text{consumioSinPromo?}(P_0, a, i)\}$

**Post**  $\equiv \{p =_{obs} olvidarÍtem(P_0, a, i)\}$

**Complejidad**:  $O(\log(A) + \log(I))$  ✓

**Descripción**: Dados un puesto, persona e ítem, borra un venta que involucra a los tres.

**Aliasing**: No genera aliasing.

CONSUMIÓSINPROMO?(in  $p$ : puesto, in  $a$ : personaId, in  $i$ : ítem)  $\rightarrow res$ : bool

**Pre**  $\equiv \{i \in \text{menú}(p)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{consumióSinPromo?}(p, a, i)\}$

**Complejidad**:  $O(\log(A) + \log(I))$

**Descripción**: Devuelve un valor de verdad respecto si hubo una venta sin promoción dados un puesto, persona e ítem.

**Aliasing**: No genera aliasing.

ESTÁENMENÚ?(in  $p$ : puesto, in  $i$ : ítem)  $\rightarrow res$ : bool

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

**Post**  $\equiv \{res =_{\text{obs}} i \in \text{menú}(p)\}$

**Complejidad**:  $O(\log(I))$

**Descripción**: Devuelve true si el ítem se encuentra en el menú, falso en otro caso.

**Aliasing**: No genera aliasing.

## 3.2. Representación

### Representación del Puesto De Comidas

puesto se representa con estr

donde estr es  $\text{tupla}(\text{menu: diccLog}(\text{ítemId}, \text{nat}),$   
 $\text{stock: diccLog}(\text{ítemId}, \text{nat}),$   
 $\text{promociones: diccLog}(\text{ítemId}, \text{infoPromos}),$   
 $\text{gastos: diccLog}(\text{personaId}, \text{nat}),$   
 $\text{ventas: diccLog}(\text{personaId}, \text{diccLog}(\text{ítem}, \text{infoVentas}))$ )

donde infoPromo es  $\text{tupla}(\text{minN: nat},$   
 $\text{maxN: nat},$   
 $\text{descuentos: diccDig}(\text{cantidad}, \text{descuento}))$

donde infoVentas es  $\text{tupla}(\text{ventasConDescuento: listaEnlazada}(\text{cant})$   
 $,$   
 $\text{ventasSinDescuento: listaEnlazada}(\text{cant}))$

Rep : estr  $\rightarrow$  bool

Rep( $e$ )  $\equiv \text{true} \iff 1 \wedge_L 2 \wedge_L 3 \wedge_L 4 \wedge_L 5 \wedge_L 6 \wedge_L 7 \wedge_L 8$

1.  $\text{claves}(e.\text{menu}) = \text{claves}(e.\text{stock})$
2.  $\text{claves}(e.\text{gastos}) = \text{claves}(e.\text{ventas})$
3.  $\text{claves}(e.\text{promociones}) \subset \text{claves}(e.\text{menu})$
4.  $(\forall i : \text{ítem})(\text{def?}(i, e.\text{promociones}) \Rightarrow_L$   
 $(\text{def?}(\text{obtener}(i, e.\text{promociones}).\text{minN}, \text{obtener}(i, e.\text{promociones}).\text{descuentos}) \wedge$   
 $\text{def?}(\text{obtener}(i, e.\text{promociones}).\text{maxN}, \text{obtener}(i, e.\text{promociones}).\text{descuentos}) \wedge_L$   
 $\neg (\exists o : \text{cant})(\text{def?}(o, \text{obtener}(i, e.\text{promociones})) \wedge (o < \text{obtener}(i, e.\text{promociones}).\text{minN} \vee$   
 $o > \text{obtener}(i, e.\text{promociones}).\text{maxN}))))))$
5.  $(\forall i : \text{ítem})(\text{def?}(i, e.\text{promociones}) \Rightarrow_L$   
 $(\forall k : \text{nat})(\text{obtener}(i, e.\text{promociones}).\text{minN} \leq k \leq \text{obtener}(i, e.\text{promociones}).\text{maxN} \Rightarrow$   
 $\text{def?}(k, \text{obtener}(i, e.\text{promociones}).\text{descuentos}))$
6.  $(\forall a : \text{persona})(\forall i : \text{ítem}) ((a \in \text{claves}(e.\text{ventas}) \wedge_L \text{def?}(i, \text{obtener}(a, e.\text{ventas})) \Rightarrow_L \text{def?}(i, e.\text{menu})))$
7.  $(\forall a : \text{persona})(\forall i : \text{ítem})(a \in \text{claves}(e.\text{ventas}) \wedge_L \text{def?}(i, \text{obtener}(a, e.\text{ventas})) \Rightarrow_L$   
 $(\forall \text{cant}_1 : \text{nat})(\text{esta?}(\text{cant}_1, \text{obtener}(i, \text{obtener}(a, e.\text{ventas})).\text{ventasConDescuento}) \Rightarrow_L$   
 $\text{cant}_1 > 0 \wedge \text{cant}_1 \geq \text{obtener}(i, \text{obtener}(a, e.\text{ventas})).\text{minN}) \wedge$   
 $(\forall \text{cant}_2 : \text{nat})(\text{esta?}(\text{cant}_2, \text{obtener}(i, \text{obtener}(a, e.\text{ventas})).\text{ventasSinDescuento}) \Rightarrow_L$   
 $\text{cant}_2 > 0 \wedge \text{cant}_2 < \text{obtener}(i, \text{obtener}(a, e.\text{ventas})).\text{minN}))$
8.  $(\forall a : \text{persona})(a \in \text{claves}(e.\text{gastos}) \Rightarrow_L$   
 $(\text{obtener}(a, e.\text{gastos}) = \text{gastoTotal}(e.\text{menu}, \text{obtener}(a, e.\text{ventas}), e.\text{promociones})))$

$\text{gastoTotal} : \{\text{dicc}(\text{ítem}, \text{nat}) \text{ precios} \times \text{dicc}(\text{ítem}, \text{tupla}(\text{secu}(\text{nat}), \text{secu}(\text{nat}))) \text{ ventas} \times \rightarrow \text{nat}$   
 $\text{dicc}(\text{ítem}, \text{tupla}(\text{nat}, \text{nat}, \text{dicc}(\text{nat}, \text{nat}))) \text{ info}\}$   
 $\{\text{claves}(\text{ventas}) \subset \text{claves}(\text{precios}) \wedge \text{claves}(\text{info}) = \text{claves}(\text{precios})\}$

```

gastoTotal(precios, venta, info)  $\equiv$  if ventas = vacío then 0
                                else if def?(dameUno(claves(ventas)), info)
                                then gastoSinDescuento(obtener(dameUno(claves(ventas)), precios),
                                 $\pi_2$  (obtener(dameUno(claves(ventas)), ventas))) +
                                gastoConDescuento(obtener(dameUno(claves(ventas)), precios),
                                 $\pi_3$ (obtener(dameUno(claves(ventas))), info),
                                 $\pi_1$  (obtener(dameUno(claves(ventas)), ventas))) +
                                gastoTotal(precios, sinUnaClave(ventas), info)
                                else gastoSinDescuento(obtener(dameUno(claves(ventas)), precios),
                                 $\pi_2$ (obtener(dameUno(claves(ventas)), ventas))) +
                                gastoTotal(precios, sinUnaClave(ventas), info)
                                fi
fi

gastoSinDescuento : nat precio  $\times$  secu(nat) s  $\rightarrow$  nat
gastoSinDescuento(precio, s)  $\equiv$  if Vacía?(s) then 0 else prim(s)  $\times$  precio + gastoSinDescuento(precio, fin(s)) fi

gastoConDescuento : nat precio  $\times$  dicc(nat, nat) d  $\times$  secu(nat) s  $\rightarrow$  nat
                                                                 $\{(\forall n: \text{nat})(\text{Está?}(n, s) \Rightarrow \text{def?}(n, d))\}$ 
gastoConDescuento(precio, d, s)  $\equiv$  if Vacía?(s) then 0 else aplicarDescuento(precio  $\times$  prim(s), obtener(prim(s), d))
+ gastoConDescuento(precio, d, fin(s)) fi

Abs : estr e  $\rightarrow$  puesto
Abs(e)  $\equiv$  p : puesto /
    menu(p) = claves(e.menu)  $\wedge_L$  ( $\forall i : \text{item}$ )( $i \in \text{menu}(p) \Rightarrow_L$ 
    precio(p, i) = obtener(i, e.menu)  $\wedge$  stock(p, i) = obtener(i, e.stock)  $\wedge$  ( $\neg \text{def?}(i, \text{e.promociones}) \Rightarrow$ 
    ( $\forall c : \text{cant}$ )(descuento(p, i, c) = 0))  $\wedge$  ( $\text{def?}(i, \text{e.promociones}) \Rightarrow$ 
    ( $\forall c : \text{cant}$ )( $(c < \text{obtener}(i, \text{e.promociones}).\text{mínN} \text{ descuento}(p, i, c) = 0) \vee$ 
    ( $\text{obtener}(i, \text{e.promociones}).\text{mínN} \leq c \leq \text{obtener}(i, \text{e.promociones}).\text{máxN} \wedge$ 
    descuento(p, i, c) = obtener(c, obtener(i, e.promociones).descuentos))  $\vee$ 
    ( $c > \text{obtener}(i, \text{e.promociones}).\text{máxN} \wedge \text{descuento}(p, i, c) = \text{obtener}(\text{obtener}(i, \text{e.promociones}).\text{máxN},$ 
    obtener(i, e.promociones).descuentos)))  $\wedge_L$ 
    ( $\forall a : \text{persona}$ )( $(\neg \text{def?}(a, \text{e.personas}) \Rightarrow_L$ 
    ventas(p, a) =  $\emptyset$ )  $\wedge$  ( $\text{def?}(a, \text{e.personas}) \Rightarrow_L$ 
    ventas(p, a) = unirVentas(obtener(a, e.ventas))))
    unirVentas : dicc(item,  $\langle \text{secu}(\text{nat}), \text{secu}(\text{nat}) \rangle$ ) d  $\rightarrow$  multiconj( $\langle \text{item}, \text{nat} \rangle$ )
    unirVentas(d)  $\equiv$  if d =obs vacío() then  $\emptyset$ 
                    else SecuAMulticonj(dameUno(claves(d)),  $\pi_1$ (obtener(dameUno(claves(d)), d)))  $\cup$ 
                    SecuAMulticonj(dameUno(claves(d)),  $\pi_2$ (obtener(dameUno(claves(d)), d)))  $\cup$ 
                    unirVentas(sinUnaClave(d))
                    fi
    SecuAMulticonj : item i  $\times$  secu(nat) s  $\rightarrow$  multiconj( $\langle \text{item}, \text{nat} \rangle$ )
    SecuAMulticonj(i, s)  $\equiv$  if Vacía?(s) then  $\emptyset$  else Ag( $\langle i, \text{prim}(s) \rangle$ , SecuAMulticonj(i, fin(s))) fi

```

### 3.3. Algoritmos

#### Algoritmos del módulo

---



---

**iCrearPuesto**(in *menu*: diccLog(item, nat), in *stock*: diccLog(item, nat), in *promos*: diccLog(item, diccLog(nat, nat))) → *res* : puesto

```

1: MenuCopia ← copiar(menu)                                ▷ O(I)
2: StockCopia ← copiar(stock)                              ▷ O(I)
3: Gastos ← VACIO()                                       ▷ O(1)
4: Ventas ← VACIO()                                       ▷ O(1)
5: promociones ← VACIO()                                  ▷ O(1)
6: it1 ← crearIt(promos)                                   ▷ O(1)
7: while HaySiguiente(it1) do                               ▷ I veces el primer while; I × O((D × log(N) + N × log(N)))
8:   item ← Siguiente(it1)                                ▷ O(1)
9:   descuentos ← VACIO()                                  ▷ O(1)
10:  It2 ← crearIt(obtener(promos, item))                  ▷ O(1)
11:  minN ← SiguienteClave(it2)                            ▷ O(1)
12:  maxN ← 0                                              ▷ O(1)
13:  while HaySiguiente(it2) do                             ▷ costo de ciclo: D veces × O(log(N))
14:    definir(descuentos, SiguienteClave(it2), SiguienteSignificado(it2)) ✓ ▷ O(log(N))
15:    maxN ← SiguienteClave(it2)                          ▷ O(1)
16:    Avanzar(it2)                                         ▷ O(1)
17:  end while
18:  ultimoDescuento ← obtener(descuentos, minN)           ▷ O(1)
19:  for int j = minN; j <= maxN; j ++ do ✓                ▷ N veces como mucho; O(N × log(N))
20:    if !definido?(descuentos, j)
21:      then definir(descuentos, j, ultimoDescuento)      ▷ diccDigital; O(log(N))
22:      else ultimoDescuento ← obtener(descuentos, j)
23:    fi
24:  end for
25:  definir(promociones, item, ⟨mnN, mxN, descuentos⟩)    ▷ O(log(I))
26:  Avanzar(it1)
27: end while
28: res ← ⟨MenuCopia, StockCopia, promociones, Gastos, Ventas⟩

```

Complejidad:  $O(I \times (D \times \log(N) + N \times \log(N)))$

Justificación: [\*] N es el máxima cantidad para la que esta definida un descuento. D es el máximo cardinal de los descuentos pasados por parámetro, definidos para un ítem. ✓

---



---



---

**iObtenerDescuento**(in *p*: estr, in *i*: ítem, in *c*: nat) → *res* : nat

```

1: res ← 0                                                ▷ O(1)
2: if definido?(p.promociones, i)                        ▷ O(log(I))
3:   then infoP ← obtener(p.promociones, i)              ▷ O(log(I))
4:     if infoPromo → maxN ≤ cant                          ▷ O(1)
5:       then res ← obtener(infoPromo → descuentos, infoPromo → maxN) ✓ ▷ O(log(cant))
6:       else if infoPromo → minN ≤ cant                  ▷ O(1)
7:         then res ← obtener(infoPromo → descuentos, infoPromo → cant) ▷ O(log(cant))
8:       fi
9:     fi
10: fi

```

Complejidad:  $O(\log(I) + \log(cant))$

---



---



---

**iObtenerGasto**(in *p*: estr, in *a*: personaId) → *res* : nat

```

1: gasto ← obtener(p.gastos, a)                          ▷ O(log(A))
2: res ← copiar(gasto)                                   ▷ O(1)

```

Complejidad:  $O(\log(A))$  ✓

---

---

**iVender**(in/out  $p$ : estr, in  $a$ : personaId, in  $i$ : ítem, in  $c$ : nat)  $\rightarrow res$ : nat

---

1:  $precio \leftarrow significado(p.menu, i)$   $\triangleright O(\log(I))$   
2:  $precioSinDescuento \leftarrow precio * cant$   $\triangleright O(1)$  ✓  
3:  $desc \leftarrow obtenerDescuento(p, i, cant)$   $\triangleright O(\log(I) + \log(cant))$  ✓  
4:  $gasto \leftarrow aplicarDescuento(precioSinDescuento, desc)$   $\triangleright O(1)$   
5: **if**  $\neg definido?(p.gastos, a)$  **then**  $\triangleright O(\log(A))$   
6:    $definir(p.gastos, a, 0)$   $\triangleright O(\log(A))$   
7:  $gastoActual \leftarrow significado(p.gastos, a)$   $\triangleright$  Referencia modificable;  $O(\log(A))$  ✓  
8:  $gastoActual \leftarrow gastoActual + gasto$   $\triangleright O(1)$   
9:  $stock \leftarrow significado(p.stock, i)$   $\triangleright$  Referencia modificable;  $O(\log(I))$  ✓  
10:  $stock \leftarrow stock - cant$   $\triangleright O(1)$   
11: //Actualizo historial.  
12: **if**  $\neg definido?(p.ventas, a)$  **then**  $\triangleright O(\log(A))$  ✓  
13:    $definir(p.ventas, a, Vacio())$   $\triangleright O(\log(A))$   
14:  $historial \leftarrow significado(p.ventas, a)$   $\triangleright O(\log(A))$   
15: **if**  $\neg definido?(historial, i)$   $\triangleright O(\log(I))$  ✓  
16:   **then**  $definir(historial, i, tupla(VACÍA, VACÍA))$   $\triangleright O(\log(I))$   
17: **fi**  
18:  $historial\_item \leftarrow significado(historial, i)$   $\triangleright O(\log(I))$   
19: **if**  $desc = 0$   
20:   **then**  $AgregarAdelante(historial\_item.ventasSinDescuento, cant)$   $\triangleright O(1)$  ✓  
21:   **else**  $AgregarAdelante(historial\_item.ventasConDescuento, cant)$   $\triangleright O(1)$   
22: **fi**  
23:  $res \leftarrow gasto$   
Complejidad:  $O(\log(A) + \log(I) + \log(cant))$  ✓

---



---

**iOlvidarItem**(in  $p$ : estr, in  $a$ : personaId, in  $i$ : ítem)  $\rightarrow res$ : nat

---

1:  $precio \leftarrow significado(p.menu, i)$   $\triangleright O(\log(I))$  // ✓  
2:  $gasto \leftarrow significado(p.gastos, a)$   $\triangleright$  Referencia modificable;  $O(\log(A))$  ✓  
3:  $gasto \leftarrow gasto - precio$   
4:  $stock \leftarrow significado(p.stock, i)$   $\triangleright$  Referencia modificable;  $O(\log(I))$  ✓  
5:  $stock \leftarrow stock + 1$   $\triangleright O(1)$  ✓  
6:  $historial \leftarrow significado(p.ventas, a)$   $\triangleright O(\log(A))$  ✓  
7:  $historial\_item \leftarrow significado(historial, i)$   $\triangleright O(\log(I))$  ✓  
8:  $cantidadConsumida \leftarrow primero(historial\_item.ventasSinDescuento)$   $\triangleright$  Referencia modificable  
9: **if**  $cantidadConsumida = 1$   
10:   **then**  $fin(historial\_item.ventasSinDescuento)$   $\triangleright O(1)$  ✓  
11:   **else**  $cantidadConsumida \leftarrow cantidadConsumida - 1$   $\triangleright O(1)$   
12: **fi**  
Complejidad:  $O(\log(A) + \log(I))$

---



---

**iConsumioSinPromo?**(in  $p$ : estr, in  $a$ : personaId, in  $i$ : ítem)  $\rightarrow res$ : bool

---

1:  $historialVentas \leftarrow significado(p.ventas, a)$   $\triangleright O(\log(A))$   
2:  $ventasSinDesc \leftarrow significado(historialVentas, i).ventasSinDescuento$  ✓  $\triangleright O(\log(I))$   
3:  $res \leftarrow EsVacia?(ventasSinDesc)$   $\triangleright O(1)$   
Complejidad:  $O(\log(A) + \log(I))$

---



---

**iPrecio**(in  $p$ : estr, in  $i$ : ítem)  $\rightarrow res$ : nat

---

1:  $res \leftarrow significado(p.menú, i)$  ✓  $\triangleright O(\log(I))$   
Complejidad:  $O(\log(I))$

---

---

---

**iObtenerStock**(in  $p$ : **estr**, in  $i$ : **ítem**)  $\rightarrow res$  : nat

1:  $res \leftarrow significado(p.stock, i)$

$\triangleright O(\log(I))$

Complejidad:  $O(\log(I))$

---

---

---

**EstaEnMenu?**(in  $p$ : **estr**, in  $i$ : **ítem**)  $\rightarrow res$  : bool

1:  $res \leftarrow definido?(p.menú, i)$

$\triangleright O(\log(I))$

Complejidad:  $O(\log(I))$

---



## 4. Diccionario Digital

### 4.1. Interfaz

parámetros formales

géneros  $\sigma$

función  $\text{COPIAR}(\text{in } s : \alpha) \rightarrow \text{res} : \sigma$

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

**Post**  $\equiv \{\text{res} =_{\text{obs}} s\}$

**Complejidad:**  $\Theta(\text{copy}(s))$

**Descripción:** función de copia de  $\sigma$ 's  $\text{COPIAR}(\text{in } a : \alpha) \rightarrow \text{res} : \alpha$  función de copia, con costo temporal  $\Theta(\text{copy}(a))$ .

se explica con:  $\text{DICCIONARIO}(\text{NAT}, \sigma)$ .

géneros:  $\text{diccDig}(\text{nat}, \sigma)$ .

### Operaciones básicas de Diccionario Digital

$\text{VACÍO}() \rightarrow \text{res} : \text{diccDig}(\text{nat}, \sigma)$

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

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

**Complejidad:**  $\Theta(1)$

**Descripción:** Crea diccionario vacío.

**Aliasing:** No genera aliasing.

$\text{DEFINIR}(\text{in/out } d : \text{diccDig}(\text{nat}, \sigma), \text{in } k : \text{nat}, \text{in } s : \sigma)$

**Pre**  $\equiv \{d = D_0\}$

**Post**  $\equiv \{d =_{\text{obs}} \text{definir}(k, s, D_0)\}$

**Complejidad:**  $O(\log(k))$

**Descripción:** Definimos s en la clave k

**Aliasing:** Define una referencia a s en el diccionario.

$\text{DEFINIDO?}(\text{in/out } d : \text{diccDig}(\text{nat}, \sigma), \text{in } k : \text{nat}) \rightarrow \text{res} : \text{bool}$

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

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

**Complejidad:**  $O(\log(k))$

**Descripción:** Revisa si k está definido.

**Aliasing:** No genera aliasing.

$\text{SIGNIFICADO}(\text{in/out } d : \text{diccDig}(\text{nat}, \sigma), \text{in } k : \text{nat}, \text{in } s : \sigma) \rightarrow \text{res} : \sigma$

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

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

**Complejidad:**  $O(\log(k))$

**Aliasing:** Devuelve una referencia modificable.

### 4.2. Representación

#### Representación del Diccionario Digital

$\text{diccDig}$  se representa con estr

donde estr es  $\text{tupla}(\text{raíz: puntero (nodo)})$

donde nodo es  $\text{tupla}(\text{hijos: arreglo\_estático}[10] \text{ de puntero(nodo)}, \text{valor: puntero}(\sigma))$

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

$\text{Rep}(e) \equiv \text{true} \iff (\text{raíz} = \text{NULL} \vee_{\text{L}} (\text{raíz}[0] \neq \text{NULL}) \wedge \text{invarianteDigital}(*\text{raíz}))$

$\text{Abs} : \text{estr } e \rightarrow \text{diccDig}(\text{nat}, \sigma)$

$\text{Abs}(e) \equiv d : \text{diccDig}(\text{nat}, \sigma) / (\forall k : \text{nat})(\text{estáDefinida?}(k, *e.\text{raíz}, \text{dígitos}(k)) \iff \text{def?}(j, d) \wedge_{\text{L}} (\forall h : \text{nat})(\text{def?}(h, d) \Rightarrow_{\text{L}} \text{significado}(h, *e.\text{raíz}) =_{\text{obs}} \text{obtener}(h, d)))$

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

$\text{invarianteDigital} : \text{nodo } n \times \text{secu}(\text{puntero}(\text{nodo})) \text{ hijos} \rightarrow \text{bool}$

*faltó definir*

$\text{invarianteDigital}(n, \text{hijos}) \equiv n.\text{valor} = \text{NULL} \Rightarrow \text{tieneHijos}(n) \wedge \text{invarianteSobreHijos}(n.\text{hijos})$

$\text{invarianteSobreHijos} : \text{secu}(\text{puntero}(\text{nodo})) \ s \rightarrow \text{bool}$

$\text{invarianteSobreHijos}(s) \equiv \text{if } \neg \text{vacía}(s) \text{ then } (\text{prim}(s) = \text{NULL} \vee_L \text{invarianteDigital}(*\text{prim}(s), \text{prim}(s) \rightarrow \text{hijos})) \wedge \text{invarianteSobreHijos}(\text{fin}(s)) \text{ else true}$  ✓

$\text{estáDefinida?} : \text{nat } k \times \text{nodo } n \times \text{nat } c \rightarrow \text{bool}$

$\{c \geq 1\}$

$\text{estáDefinida?}(k, n, c) \equiv \text{if } c = 1 \text{ then if } n = 0 \text{ then } n.\text{valor} \neq \text{NULL} \text{ else } n\hat{\text{E}}\text{simo}(\text{primerDígito}(K), n.\text{hijos}) \neq \text{NULL} \wedge_L n\hat{\text{E}}\text{simo}(\text{primerDígito}(K), n.\text{hijos}) \rightarrow \text{valor} = \text{NULL} \text{ fi} \text{ else if } n\hat{\text{E}}\text{simo}(\text{primerDígito}(K), n.\text{hijos}) = \text{NULL} \text{ then false} \text{ else } \text{estáDefinida?}(\text{sinPrimerDígito}(k), *n\hat{\text{E}}\text{simo}(\text{primerDígito}(k), n.\text{hijos}), c-1) \text{ fi} \text{ fi}$  ✓

$\text{significado} : \text{nat } k \times \text{nodo } n \times \text{nat } c \rightarrow \text{bool}$

$\{\text{estáDefinida}(k, n)\}$

$\text{significado}(k, n, c) \equiv \text{definir}$

$\text{dígitos} : \text{nat } n \rightarrow \text{nat}$

$\text{dígitos}(n) \equiv \text{dígitosAux}(n, 1)$

$\text{dígitosAux} : \text{nat } n \times \text{nat } m \rightarrow \text{nat}$

$\text{dígitosAux}(n, m) \equiv \text{if } n \bmod 10 \wedge m = n \text{ then } m \text{ else } \text{dígitosAux}(n, m+1) \text{ fi}$  ✓

$\text{primerDígito} : \text{nat } n \rightarrow \text{nat}$

$\text{primerDígito}(n) \equiv n \text{ div } 10 \wedge \text{dígitos}(n)-1$

$\text{sinPrimerDígito} : \text{nat } n \rightarrow \text{nat}$

$\text{sinPrimerDígito}(n) \equiv n \bmod 10 \wedge \text{dígitos}(n)-1$  ✓

$\{\text{dígitos}(n) > 1\}$

$n\hat{\text{E}}\text{simo} : \text{nat } n \times \text{secu}(\alpha \ s) \rightarrow \alpha$

$\{n < \text{long}(s)\}$

$n\hat{\text{E}}\text{simo}(n, s) \equiv \text{if } n = 0 \text{ then } \text{prim}(s) \text{ else } n\hat{\text{E}}\text{simo}(n-1, \text{fin}(s)) \text{ fi}$  ✓

$\text{div} : \text{nat } n \times \text{nat } k \rightarrow \text{nat}$

$\text{div}(n, k) \equiv \text{if } n < k \text{ then } 0 \text{ else } 1 + \text{div}(n-k, k) \text{ fi}$

$\bullet \text{ mod } \bullet : \text{nat } n \times \text{nat } k \rightarrow \text{nat}$

$\{k > 1\}$

$n \bmod m \equiv n - \text{div}(n, k) \times k$

$\bullet \wedge \bullet : \text{nat } n \times \text{nat } p \rightarrow \text{nat}$

$\{n = 0 \Rightarrow k > 0\}$

$n \wedge k \equiv \text{if } k = 0 \text{ then } 1 \text{ else } n \times (n \wedge (k-1)) \text{ fi}$

### 4.3. Algoritmos

#### Algoritmos del módulo

---

---

**iVacío()**  $\rightarrow res : \text{estr}$

```
1: var AE : arreglo_estático[10] de puntero(nodo)  $\triangleright O(1)$ 
2: for int i = 0; i < 10; i ++ do  $\triangleright 10 \times O(1) = O(1)$ 
3:   AE[i]  $\leftarrow$  NULL  $\triangleright O(1)$ 
4: end for
5: res  $\leftarrow$   $\langle \& \langle AE, NULL \rangle \rangle$   $\triangleright O(1)$ 
```

Complejidad:  $O(1)$

Justificación:  $O(1) + O(1) + O(1) = O(1)$ .

---

Esta es una operación privada que recibe un natural n cualquiera y devuelve en res su cantidad de dígitos:

---

---

**iDígitos(in n : nat)**  $\rightarrow res : \text{nat}$

```
1: res  $\leftarrow$  1  $\triangleright O(1)$ 
2: while n mod  $10^{\text{res}} \neq n$  do  $\triangleright [*]$ 
3:   res ++  $\triangleright O(1)$ 
4: end while
```

Complejidad:  $O(\log(n))$

Justificación: Basta notar que  $n \bmod 10^{\text{res}} = n \iff n < 10^{\text{res}} \iff \log(n) < \text{res}$ . Luego, como res aumenta en 1 en cada iteración, en el peor caso el while se ejecuta  $\log(n)$  veces. Por álgebra de órdenes,  $O(1) + O(\log(n)) = O(\log(n))$ .

---

Esta es una operación privada que recibe un natural n cualquiera, su cantidad de dígitos  $c > 0$  y devuelve en res el dígito “más a la izquierda”:

---

---

**iPrimerDígito(in n : nat, in c : nat)**  $\rightarrow res : \text{nat}$

```
1: res  $\leftarrow$  n div  $0^{(c-1)}$   $\triangleright O(1)$ 
```

Complejidad:  $O(1)$

---

Esta es una operación privada que recibe un natural n cualquiera, su cantidad de dígitos  $c > 0$ , y devuelve otro natural res que es el resultado de ponerle en cero el primer dígito al pasado como parámetro:

---

---

**iSinPrimerDígito(in n : nat, in c : nat)**  $\rightarrow res : \text{nat}$

```
1: res  $\leftarrow$  n mod  $0^{(c-1)}$   $\triangleright O(1)$ 
```

Complejidad:  $O(1)$

---

---

---

**iDefinir(in/out d : estr, in k : nat, in s :  $\sigma$ )**

```
1: nat i  $\leftarrow$  Dígitos(k)  $\triangleright O(\log(k))$ 
2: puntero(nodo) actual  $\leftarrow$  estr.raíz  $\triangleright O(1)$ 
3: while i > 0 do  $\triangleright [*]$ 
4:   if actual  $\rightarrow$  hijos[PrimerDígito(k, i)] = NULL  $\triangleright O(1)$ 
5:     then var AE : arreglo_estático[10] de puntero(nodo)  $\triangleright O(1)$ 
6:     for int j = 0; j < 10; j ++ do  $\triangleright 10 \times O(1) = O(1)$ 
7:       AE[j]  $\leftarrow$  NULL  $\triangleright O(1)$ 
8:     end for
9:     actual  $\rightarrow$  hijos[PrimerDígito(k, i)]  $\leftarrow$   $\& \langle AE, NULL \rangle$   $\triangleright O(1)$ 
10:    actual  $\leftarrow$  actual  $\rightarrow$  hijos[PrimerDígito(k, i)]  $\triangleright O(1)$ 
11:    k  $\leftarrow$  sinPrimerDígito(k, i)  $\triangleright O(1)$ 
12:    i --  $\triangleright O(1)$ 
13: end while
14: actual  $\rightarrow$  valor  $\leftarrow$   $\&s$   $\triangleright O(1)$ 
```

Complejidad:  $O(\log(k))$   $[*]$

Justificación: El while se ejecuta  $\log(k)$  veces en el peor caso, por el mismo argumento de la complejidad de Dígitos  $O(\log(k)) + O(1) + O(\log(k)) + O(1) = O(\log(k))$ .

---

---

---

**iDefinido?**(in  $d$ : **estr**, in  $k$ : **nat**,)  $\rightarrow res$  : **bool**

1: $nat\ i \leftarrow D\acute{ig}itos(k)$	$\triangleright O(\log(k))$
2: $puntero(nodo)\ actual \leftarrow estr.ra\acute{ı}z$	$\triangleright O(1)$
3: <b>while</b> $i > 0 \ \&\& \ actual \rightarrow hijos[PrimerD\acute{ig}ito(k, i)] \neq NULL$ <b>do</b>	$\triangleright O(\log(k))$
4: $actual \leftarrow actual \rightarrow hijos[PrimerD\acute{ig}ito(k, i)]$	$\triangleright O(1)$
5: $k \leftarrow sinPrimerD\acute{ig}itp(k, i)$	$\triangleright O(1)$
6: $i - -$	$\triangleright O(1)$
7: <b>end while</b>	
8: $res \leftarrow (i = 0 \ \&\& \ actual \rightarrow valor \neq NULL)$	$\triangleright O(1)$

Complejidad:  $O(\log(k))$

Justificaci3n: Estructura similar a la anterior.

---

---

---

**iSignificado?**(in  $d$ : **estr**, in  $k$ : **nat**,)  $\rightarrow res$  :  $\sigma$

1: $nat\ i \leftarrow D\acute{ig}itos(k)$	$\triangleright O(\log(k))$
2: $puntero(nodo)\ actual \leftarrow estr.ra\acute{ı}z$	$\triangleright O(1)$
3: <b>while</b> $i > 0$ <b>do</b>	$\triangleright O(\log(k))$
4: $actual \leftarrow actual \rightarrow hijos[PrimerD\acute{ig}ito(k, i)]$	$\triangleright O(1)$
5: $k \leftarrow sinPrimerD\acute{ig}itp(k, i)$	$\triangleright O(1)$
6: $i - -$	$\triangleright O(1)$
7: <b>end while</b>	
8: $res \leftarrow actual \rightarrow valor$	$\triangleright O(1)$

Complejidad:  $O(\log(k))$

Justificaci3n: Estructura similar a la anterior.

---