Algoritmos y Estructuras de Datos II

Trabajo Práctico 2

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

Trabajo Práctico 2: Diseño

Lollapatuza

Integrante	LU	Correo electrónico
Marziano, Franco	849/19	franco.marziano.96@gmail.com
Gonzalez Correas, Alvaro	233/22	alvarogonzalezc4@gmail.com
Vazquez, Martin Ignacio	327/17	vazquez.martin.ignacio@gmail.com
Sturm, Candelaria	244/20	sturmcande@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Lollapatuza

Interfaz

se explica con: DICCIONARIO(κ, σ),LOLLAPATUZA,PUESTODECOMIDA,DINERO

géneros:

Operaciones básicas de Lollapatuza

CREARLOLLA(**in** ps: `dicc(puestoID,puesto)`,**in** per: `conj(persona)`) $\rightarrow$ res : `lolla`
Pre $\equiv$ {vendenAlMismoPrecio(significados(ps) $\wedge NoVendieronAun(significados(ps) \wedge \neg\emptyset?(ps) \wedge \neg\emptyset?(claves(ps))))$}
Post$\equiv$ {res=crearLolla(ps,per)}
Complejidad: $O(copy(ps) + copy(per))$
Descripción: Inicia el sistema de un festival
Aliasing: ps y per se agregan por copia

REGISTRARCOMPRA(Inout l:lolla, **in** id: `puestoID`,**in** p: `persona`,**in** i: `item`,**in** c: `cant`) $\rightarrow$ res : `lolla`
Pre $\equiv$ {l=$l_0 \wedge p \in personas(l) \wedge def?(id, puestos(l) \wedge_L haySuficiente?(obtener(id, puestos(l)), , i, c))$}
Post$\equiv$ {res = vender(l_o, id, p, i, c)}
Complejidad:$O(Log(A) + Log(I) + Log(P)$
Descripción: Registra una compra de una cantidad de un ítem, realizada por una persona en un puesto

HACKEAR(Inout l:lolla, **in** p: `persona`,**in** i: `item`) $\rightarrow$ res : `lolla`
Pre $\equiv$ {l=$l_0 \wedge ComsumioSinPromoEnAlgunPuesto(l, p, i)$}
Post$\equiv$ {res = hackear(l_0, p, i)}
Complejidad:$(O(Log(A) + Log(I))o(O(Log(A) + Log(I) + Log(P))sielpuestodejadeserhackeable$
Descripción: Hackea un item consumido por una persona del puesto con el menor ID en el que la persona haya consumido sin promoción.

GASTOTOTAL(**in** l: `lolla`, **in** p: `persona`) $\rightarrow$ res : `dinero`
Pre $\equiv$ {p$\in$ $personas(l)$}
Post$\equiv$ {res = gastoTotal(l,p)}
Complejidad: $O(log(A))$
Descripción: Devuelve el gasto total de una persona

PERSONAQUEMASGASTO(**in** l: `lolla`) $\rightarrow$ res : `persona`
Pre $\equiv$ {true}
Post $\equiv$ {res = masGasto(l)}
Complejidad: $O(1)$
Descripción: Devuelve la persona que más dinero gasto. En caso de haber varias personas con el monto máximo gastado, desempata por ID de la persona

MENORSTOCKDEITEM(**in** l: `lolla`, **in** i: `item`) $\rightarrow$ res : `puestoID`
Pre $\equiv$ {true}
Post $\equiv$ {res = menorStock(l,i)}
Complejidad: $O(P * log(I))$
Descripción: Devuelve el ID del puesto con menor stock del item. Si hay varios puestos con stock mínimo, deevuelve el de menor ID

PERSONAS(**in** l: `lolla`) $\rightarrow$ res : `conj(persona)`
Pre $\equiv$ {true}
Post $\equiv$ {res = personas(l)}
Complejidad: $O(1)$
Aliasing: no hay

PuestosDeComida(**in** l : `lolla`) → res : `dicc(puestoID:puesto)`
Pre ≡ {true}
Post ≡ {res = puestos(l)}
Complejidad: $O(1)$
Descripción: Devuelve los puestos de comida con sus IDs
Aliasing: no hay

Representación

`Lolla` se representa con `estr`

donde `estr` es `tupla`(*puestos*: `diccLog(puestoID,puesto)`
, *personas*: `conjLineal(persona)`
, *GastoTotal*: `diccLog(persona,nat)`
, *GTxPrecio*: `diccLog(nat,diccLog(persona,nat))`
, *PersonaQueMasGasto*: `persona*`
, *PuestosHackeables*: `diccLog(persona:diccLog(item:diccLog(puestoID,puesto*)))`
)

Rep : estr ⟶ bool

Rep(e) ≡ true ⟺
$(\forall id1, id2 : puestoID)(def?(id1, e.puestos) \land def?(id2, e.puestos) \land id1 \neq id2 \Longrightarrow_L obtener(id1, e.puestos) \neq obtener(id2, e.puestos)) \land$

claves(e.GastoTotal) ⊆ $e.personas \land_L$

$(\forall p : persona)(def?(p, e.GastoTotal \longrightarrow_L obtener(p, e.GastoTotal) = totalVentas(p, significados(e.puestos)))) \land$
significados(e.GastoTotal) = claves(e.GTxPrecio) $\land_L$

$(\forall p : persona)(def?(p, e.GastoTotal) \Longrightarrow_L p \in claves(obtener(obtener(p, e.GastoTotal), e.GTxPrecio))) \land_L$

claves(e.PuestosHackeables) ⊆ $e.personas \land_L$

$(\forall p : persona, i : item)(def?(p, e.PuestosHackeables) \land_L def?(i, obtener(p, e.PuestosHackeables)) \Longrightarrow_L$
$(\forall id : puestoID)(def?(id, obtener(i, obtener(p, e.PuestosHackeables))) \Longrightarrow_L$
$*obtener(id, obtener(i, obtener(p, e.PuestosHackeables))) = obtener(id, e.puestos) \land_L$
$def?(i, obtener(id, e.puestos).menu)))$

Abs : estr e ⟶ lolla {Rep(e)}

Abs(e) ≡ puestos(lolla) = e.puestos ∧
$personas(lolla) = e.personas$

Funciones auxiliares

totalVentas : p:persona × ps:conj(puesto) ⟶ nat

totalVentas(p,ps) ≡ **if** ∅?(ps) **then** 0 **else** obtener(p,dameUno(ps).GastoTotal) + totalVentas(p,sinUno(ps)) **fi**)

---

iCrearLolla(**in** ps : `dicc(puestoID, puesto)`, **in** per : `conj(persona)`) ⟶ `Lolla`

```
1: res ⟵ <puestos(ps), personas(per), GastoTotal(diccLog :: Vacio()),
3:   GTxPrecio(diccLog::Vacio()),PuestosHackeables(diccLog::Vacio()),PersonaQueMasGasto(NULL)⟩
4:   devolver res                                          ▷ O(copy(ps)+copy(per))
```

---

iRegistrarCompra(**inout** l : `lolla`, **in** p : `puestoid`, **in** *per* : `persona`,**in** i : `item`,**in** *cant* : `nat`)

```
1:  pu* ← &(significado(l.puestos, p))                                                    ▷ O(log(P)
2:  precio ← significado((*pu).menu, i)                                                  ▷ O(log(I)
3:  Desc ← iObtenerDescuento((*pu), i, c)                                                 ▷ O(log(I))
4:  Gasto ← (precio * c) * (1 − (Desc/100))
5:  iVender((*pu),per,i,c)                                                                ▷ O(log(I)+log(A))
6:  si definido(l.GastoTotal,per) entonces                                                ▷ O(log(A))
7:      gastoAnterior ← significado(l.GastoTotal, per)
8:      definir(l.GastoTotal,per,gastoAnterior+Gasto)                                     ▷ O(log(A))
9:  else
10:     Definir(l.GastoTotal,per,gasto)                                                   ▷ O(log(A))
11:     gastoAnterior ← 0
12: si definido(l.GTxPrecio,gastoAnterior) entonces                                       ▷ O(log(A))
13:     borrar(significado(l.GTxPuesto,gastoAnterior),per)                                ▷ O(2log(A))
14:     si significado(l.GTxPuesto,gastoAnterior) == Vacio() entonces                     ▷ O(log(A))
15:         borrar(l.GTxPuesto,gastoAnterior)                                             ▷ O(log(A))
16:     else
17: else
18:
19:
20: si definido(l.GTxPrecio,gastoAnterior+Gasto) entonces                                 ▷ O(log(A))
21:     definir(l.GTxPrecio,gastoAnterior+Gasto,definir(significado(l.GTxPrecio,gastoAnterior+Gasto),per,0))   ▷
    O(2log(A))
22: else
23:     definir(l.GTxPrecio,gastoAnterior+Gasto,definir(Vacio(),per,0))                   ▷ O(log(A))
24:
25: si Desc == 0 entonces
26:
27:     si definido(l.PuestosHackeables,per) entonces                                     ▷ O(log(A))
28:
29:         si definido(significado(l.PuestosHackeables,per),i) entonces                  ▷ O(log(A) + log(I))
30:
31:             si !definido(significado(significado(l.PuestosHackeables,per),i),p)) entonces   ▷
    O(log(A)+log(I)+log(P))
32:                 Definir(significado(significado(l.PuestosHackeables,per),i),p,pu)     ▷ O(log(A)+log(I)+log(P))
33:             else
34:         else  definir(l.PuestosHackeables,i,definir(Vacio(),p,pu))                    ▷ O(log(A) + log(I))
35:     else  definir(l.PuestosHackeables,per,definir(Vacio(),i,definir(Vacio(),p,pu)))   ▷ O(log(A))
36: else
37:
38: it = CrearIt(l.GTxPrecio)          ▷ aca asumimos que crearIt te da un iterador a la clave maxima   ▷ O(1)
39: it2 = CrearIt(siguienteSignificado(it))  ▷ aca asumimos que crearIt te da un iterador a la minima clave ya que es
    otro diccLog distinto                                                                 ▷ O(log(1)
40: l.personaQueMasGasto ← &siguienteClave(it2)                                           ▷ O(1)
```

GastoTotal(**in** l : `lolla`, **in** p : `persona`) ⟶ `nat`

```
1: si definido(l.GastoTotal,p) entonces                                                   ▷ O(log(A))
2:     devolver significado(l.GastoTotal,p)                                               ▷ O(log(A))
3: else
4:     devolver 0
```

PersonaQueMasGasto(**in** l : `lolla`) $\longrightarrow$ `persona`

```
1: si l.PersonaQueMasGasto = NULL entonces                    ▷ O(1)
2:     it = CrearIt(l.personas)                               ▷ O(1)
3:     devolver siguienteClave(it)                            ▷ O(1)
4: else
5:     devolver *(l.PersonaQueMasGasto)                       ▷ O(1)
```

iMenorStockDeItem(**in** l : `lolla`, **in** i : `item`) $\longrightarrow$ `puestoID`

```
1: it ← CrearIt(e.puestos)                                    ▷ esto es O(1)
2: min ← SiguienteClave(it)                                   ▷ O(1)
3: mientras haySiguiente(it)                                  ▷ O(P)
4: hacer {
5: stockit ← iStock(SiguienteSignificado(it), i)
6: stockmin ← iStock(significado(l.puestos, min), i)
7: si (stockit <stockmin) || (stockit = stockmin && SiguienteClave(it) <min) entonces   ▷ O(2log(I))
8:     min = SiguienteClave(it)                               ▷ O(1)
9:     Avanzar(it)                                            ▷ O(1)
10: else
11:    Avanzar(it)                                            ▷ O(1)
   }
12:    devolver MIN
```

iHackear(**inout** l : `lolla`, **in** *per* : `persona`,**in** i : `item`)

```
1: it = CrearIt(significado(significado(l.PuestosHackeables,per),i)) ▷ CrearIt nos da el puesto hackeable de menor id O(log(A)+log(I))
2: pu* = siguienteSignificado(it) ▷ O(1)
3: it2 = primero(significado(significado(*pu.ventasSinDescuento,per),i)) ▷ O(1 + log(A) + log(I) + 1)
4: stockAnt ← significado(∗pu.stock, i) ▷ O(log(I))
5: definir(*pu.stock,i,stockAnt + 1) ▷ O(log(I))
6: precio ← significado(∗pu.menu, i) ▷ O(log(I))
7: gastoPuestoAnt ← significado(∗pu.GastoxPersona, per) ▷ O(log(A))
8: si gastoPuestoAnt == precio entonces
9:     borrar(*pu.GastoxPersona,per) ▷ O(log(A))
10: else
11:     definir(*pu.GastoxPersona,per, gastoPuestoAnt - precio) ▷ O(log(A))
12: si π2(siguiente(it2)) == 1 entonces
13:     eliminarSiguiente(it2) ▷ O(1)
14:     definir(*pu.ventasSinDescuento,per,definir(significado(*pu.ventasSinDescuento,per),i,
15:     fin(significado(significado(*pu.ventasSinDescuento,per)))) ▷ Aca eliminamos it2 de la lista de itLista O(log(A) + log(I))
16: else
17:     AgregarComoSiguiente(it2,⟨i,π2(siguiente(it2))-1⟩) ▷ O(1)
18: gastoAnterior ← significado(l.GastoTotal, per) ▷ O(log(A))
19: si significado(l.GastoTotal,per) == precio entonces ▷ O(log(A))
20:     borrar(l.GastoTotal,per) ▷ O(log(A))
21: else
22:     definir(l.GastoTotal,per,gastoAnterior-precio) ▷ O(log(A))
23: borrar(significado(l.GTxPuesto,gastoAnterior),per) ▷ O(log(A))
24: si significado(l.GTxPuesto,gastoAnterior) == Vacio() entonces ▷ O(log(A))
25:     borrar(l.GTxPuesto,gastoAnterior) ▷ O(log(A))
26: else
27:
28: si definido(l.GTxPrecio,gastoAnterior-precio) entonces ▷ O(log(A))
29:     definir(l.GTxPrecio,gastoAnterior-precio,definir(significado(l.GTxPrecio,gastoAnterior-precio),per,0)) ▷ O(2log(A))
30: else
31:     definir(l.GTxPrecio,gastoAnterior-precio,definir(Vacio(),per,0)) ▷ O(log(A))
32:
33: si per == *(l.PersonaQueMasGasto) entonces ▷ O(1)
34:     itPrecio = CrearIt(l.GTxPrecio)  ▷ aca asumimos que crearIt te da un iterador a la clave maxima  ▷ O(1)
35:     itPersona = CrearIt(siguienteSignificado(itPrecio)) ▷ aca asumimos que crearIt te da un iterador a la minima clave ya que es otro diccLog distinto ▷ O(1)
36:     l.personaQueMasGasto ← &siguienteClave(itPersona) ▷ O(1)
37: else
38:
39: si significado(significado(*pu.ventasSinDescuento,per),i) == Vacia() entonces ▷ O(log(A) + log(I))
40:     definir(l.puestosHackeables,per,definir(significado(l.puestosHackeables,per),
41:     i,borrar(significado((significado(l.puestosHackeables,per),i)),siguienteClave(it)))) ▷ O(log(A) + log(I) + log(P) )
42: else
43:
44: si significado(significado(l.puestosHackeables,per),i) == Vacio() entonces ▷ O(log(A) + log(I))
45:     definir(l.puestosHackeables,per,borrar(significado(l.puestosHackeables,per),i)) ▷ O(log(A) + log(I))
46: else
47:
48: si significado(l.puestosHackeables,per) == Vacio() entonces ▷ O(log(A))
49:     borrar(l.puestosHackeables,per) ▷ O(log(A))
50: else
```

iPersonas(**in** l : `lolla`) $\longrightarrow$ `conj(persona)`	
devolver l.personas	▷ $\mathcal{O}(1)$

iPuestosDeComida(**in** l : `lolla`) $\longrightarrow$ `diccLog(puestoID:puesto)`	
1: **devolver** l.puestos	▷ $\mathcal{O}(1)$

2. Puesto de Comida

Interfaz

se explica con: Diccionario(κ, σ),PuestoDeComida,Dinero

géneros: `dicc`(κ, σ)`,Lista`(α).

Operaciones básicas de Puesto De Comida

CrearPuesto(**in** *menu* : `dicc(item,nat)`,**in** *stock* : `dicc(item,nat)`,**in** *desc* : `dicc(item,vector<nat>)`) $\rightarrow$ *res* : `puesto`
Pre $\equiv$ {claves(p) $=_{\text{obs}}$ claves(s) $\wedge$ claves(d) $\subseteq$ claves(p)}
Post $\equiv$ {res $=_{\text{obs}}$ crearPuesto(menu,stock,DiccADiccDicc(desc))}
Complejidad: $O(copy(stock) + copy(menu) + copy(desc))$
Descripción: genera un puesto vacío.
Aliasing: menu,stock y desc se agregan por copia

Stock(**in** P : `puesto`,**in** I : `item`) $\rightarrow$ *res* : `Nat`
Pre $\equiv$ {definido?(d, k)}
Post $\equiv$ {$res =_{\text{obs}}$ Significado(P, I)}
Complejidad: $O(Log(I))$
Descripción: Devuelve el Stock del item .

ObtenerDescuento(**in** p : `puesto`, **in** it : `item`, **in** c : `cant`) $\rightarrow$ *res* : `Nat`
Pre $\equiv$ {$it \in menu(p)$ }
Post $\equiv$ {res $=_{\text{obs}}$ descuento(p,it,c)}
Complejidad: $(O(Log(I))$
Descripción: Devuelve cuanto descuento tiene el item it con la cantidad c en el puesto p

Vender(Inout p:puesto, **in** *per* : `persona`,**in** i : `item`,**in** c : `cant`)
Pre $\equiv$ {i $\in menu(p) \wedge c \geq stock(p, i) \wedge p = p_0$}
Post$\equiv$ {p = vender(p_0, per, i, c)}
Complejidad:$(O(Log(I) + log(A))$
Descripción: Registra una venta en el puesto p

GastoPorPersona(**in** p : `puesto`, **in** *per* : `persona`) $\rightarrow$ *res* : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {res = gastosDe(p,per)}
Complejidad: $\Theta(log(A)$
Descripción: devuelve cuanto gastó una persona per en el puesto p

Especificación de las operaciones auxiliares utilizadas en la interfaz

DescuentoDe : dicc(item:vector(nat)) $\times$ item $\times$ cant $\longrightarrow$ nat

ObtenerDescuento(di,i,c) $\equiv$ **if** (c >longitud(obtener(i,di))) **then**
obtener(i,di)[longitud(obtener(i,di)) - 1]
else
obtener(i,di)[c]
fi)

significados : d:dicc(a:α) $\longrightarrow$ conj(α)

significados(d) $\equiv$ **if** vacio = d **then**
$\emptyset$
else
Ag(obtener(dameUno(claves(d)),d),significados(borrar(dameUno(claves(d)),d)))
fi

secuADict : s:secu(α) $\times$ pos:Nat $\times$ res:dicc(nat $\times$ α) $\longrightarrow$ dicc(nat, α)

secuADict(s,pos,res) $\equiv$ **if** vacia?(s) **then** res
else if prim(s) = 0 $\vee$ prim(s) $\in$ significados(res) **then**
secuADict(fin(s),pos+1,res)
else
definir(pos,prim(s),secuADict(fin(s),pos+1,res)) **fi**
fi

PlataGastada : secu(tupla(item,cant)) $\times$ dicc(item:vector<nat>) $\longrightarrow$ nat

PlataGastada(s,di) $\equiv$ **if** vacia?(s) **then**
0
else
aplicarDescuento(π_2(prim(s)) $*$ obtener(π_1(prim(s)),pu.menu),
DescuentoDe(di,π_1(prim(s)),π_2(prim(s)))) + plataGastada(fin(s),di)
fi

apariciones : secu(α) $\times$ α $\longrightarrow$ nat

apariciones(s,i) $\equiv$ **if** vacia?(s) **then** 0
else if prim(s) = i **then** 1 + apariciones(fin(s),i) **else** apariciones(fin(s),i) **fi** **fi**

DiccADiccDicc : dicc(item:vector(nat)) di $\times$ conj(item) ci $\times$ $\longrightarrow$ dicc(item:dicc(cant:descuento))
dicc(item:vector(nat))
{ci = claves(di)}

DiccADiccDicc(di,ci,va) $\equiv$ **if** vacio?(ci) **then**
va
else
definir(dameUno(ci),secuADict(obtener(dameUno(ci),di)),
DiccADiccDicc(di,sinUno(ci),va))
fi

estanTodos : secu(itUni(β)) $\times$ secu(β) $\longrightarrow$ bool

estanTodos(s,t) $\equiv$ **if** vacio(s) **then**
true
else
if esta?(actual(prim(s)),t) **then** estanTodos(fin(s),t)
else false **fi**
fi

Representación

`Puesto` **se representa con** `pu`

donde `pu` es `tupla`(*Stock*: `diccLog(item,nat)`
, *menu*: `diccLog(item,nat)`
, *desc*: `diccLog(item,vector(nat))`
, *GastoXPersona*: `diccLog(persona,nat)`
, *Ventas*: `diccLog(persona,ListaEnlazada(tupla(item,cant)))`
, *ventasSinDesc*: `diccLog(persona:diccLog(item:ListaEnlazada(itLista)))`)

Rep : pu $\longrightarrow$ bool

Rep(p) $\equiv$ true $\iff \forall(i : item)(def?(pu.stock, i) \iff def?(pu.menu, i)) \wedge_L$

$\forall(i : item)(def?(pu.desc, i) \Longrightarrow_L def?(pu.stock, i)) \wedge_L$

$\forall(p : persona)(def?(p, pu.GastoxPersona) \iff def?(p, pu.ventas)) \wedge$

$\forall(p : persona)(def?(p, pu.GastoxPersona) \Longrightarrow_L obtener(p, pu.GastoxPersona) = plataGastada(obtener(p, pu.ventas), pu.desc)) \wedge_L$
$\forall(p : persona)(def?(pu.ventasSinDescuento, p) \Longrightarrow_L def?(pu.ventas, p)) \wedge_L$
$\forall(p : persona)(def?(pu.ventasSinDescuento, p) \Longrightarrow_L (\forall(i : item)(def?(obtener(p, pu.ventasSinDescuento)) \Longrightarrow_L estanTodos((obtener(i, obtener(p, pu.ventasSinDescuento))), obtener(p, pu.ventas)))))$

Abs : pu p $\longrightarrow$ puesto {Rep(p)}

Abs(p) $\equiv$ menu(puesto) = claves(pu.menu) $\wedge$
$\forall(i : item)(precio(puesto, i) = obtener(i, pu.menu)) \wedge$
$\forall(i : item)(stock(puesto, i) = obtener(i, pu.stock) \wedge$
$\forall(i : item, c : cant)(descuento(puesto, i, c) = DescuentoDe(pu.desc, i, c)) \wedge$
$\forall(p : persona)(ventas(puesto, p) = long(obtener(p, pu.ventas))) \wedge$
$\forall(p : persona, t : tupla(item, cant))(t \in ventas(puesto, p) \Longrightarrow_L apariciones(obtener(a, pu.ventas), t) = (t, ventas(puesto, p)))$

---

iCrearPuesto(**in** *menu* : `dicc(item, nat)`, **in** *stock* : `dicc(item, nat)`, **in** *desc* : `dicc(item, vector<nat>)`) $\longrightarrow$ `Puesto`

1: $pu.stock \leftarrow stock$ ▷ esto es $\mathcal{O}$(copy(stock))
2: $pu.menu \leftarrow menu$ ▷ $\mathcal{O}$(copy(menu))
3: $pu.desc \leftarrow desc$ ▷ $\mathcal{O}$(copy(desc))
4: $pu.gastoPorPersona \leftarrow$ Vacio() ▷ $\mathcal{O}(1)$
5: $pu.ventas \leftarrow$ Vacio() ▷ $\mathcal{O}(1)$
6: $pu.ventasSinDescuento \leftarrow$ Vacio() ▷ $\mathcal{O}(1)$
7: **devolver** pu

---

iObtenerDescuento(**in** p : `Puesto`, **in** i : `Item`, **in** c : `cant`) $\longrightarrow$ `nat`

1: vecDesc=significado(pu.desc,i) ▷ $\mathcal{O}$(log(I))
2: **si** $c \geq longitud(vecDesc) - 1$ **entonces** ▷ $\mathcal{O}(1)$
3: **devolver** vecDesc[longitud(vecDesc)-1] ▷ $\mathcal{O}(1)$
4: **else**
5: **devolver** vecDesc[c] ▷ $\mathcal{O}(1)$

---

iStock(**in** p : `Puesto`, **in** i : `Item`) $\longrightarrow$ `nat`

1: **devolver** significado(pu.stock,i) ▷ $\mathcal{O}$(log(I))

---

iVender(**inout** p: `Puesto`, **in** *per* : `Persona`,**in** i : `item`,**c** *cant* : `cantidad`)

```
1:  Precio ⟵ Significado(p.menu, i)                                   ▷ O(log I)
2:  Desc ⟵ iObtenerDescuento(p, i, c)                                 ▷ O(log I)
3:  st ← significado(pu.stock, i)                                     ▷ O(log I)
4:  definir(pu.stock,i,st-c)                                          ▷ O(log I)
5:  Gasto ⟵ (Precio ∗ c) ∗ (1 − (desc/100))                           ▷ O(1)
6:  si definido?(p.GastoxPersona,per) entonces                        ▷ O(log A)
7:      gastoPer = significado(p.GastoxPersona,per)                   ▷ O(log A)
8:      definir(p.GastoxPersona, per, gastoPer + Gasto)               ▷ O(log A)
9:      it = AgregarAtras(significado(p.ventas,per),<i,c>)            ▷ O(log A)
10:     si desc == 0 entonces
11:         AgregarAtras(significado(significado(p.ventasSinDesc,per),i),it)   ▷ O(log A + log I)
12:     else
13: else
14:     definir(p.GastoxPersona,per,gasto)                            ▷ O(log A)
15:     definir(p.ventas,per,vacia())                                 ▷ O(log A)
16:     it2 = AgregarAtras(significado(p.ventas,per),<i,c>)           ▷ O(log A)
17:     si desc == 0 entonces
18:         definir(p.VentasSinDesc,per,definir(Vacio(),i,Vacia()))   ▷ O(log(A))
19:         AgregarAtras(significado(significado(p.ventasSinDesc,per),i),it2)   ▷ O(log A + log I)
20:     else
```

iGastoPorPersona(**in** p : `Puesto`, **in** *per* : `Persona`) ⟶ `nat`

```
1:  si definido?(pu.gastoPorPersona(per)) == true entonces            ▷ O(log(A))
2:      devolver significado(pu.gastoPorPersona, per)                 ▷ O(log(A))
3:  else
4:      devolver 0                                                    ▷ O(1)
    =0
```