Algoritmos y Estructuras de Datos II

Trabajo Práctico 2

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

Lollapatuza

Y su magnífico diseño

Integrante	LU	Correo electrónico
Gardey, Juan Pablo	1495/21	jpgardey@dc.uba.ar
Fontana Walser, Florencia	1530/21	florfontana02@gmail.com
Rossi, Hernan Guido	791/21	guidorossi1996@gmail.com
Muñoz, Joaquín Eliseo	1484/21	joaquin.e.munoz@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Módulo Lollapatuza

El módulo Lollapatuza provee una estructura que permite el manejo del sistema para el reconocido festival internacional Lollapatuza. Incluye funciones de inicialización de un festival, registro de compras, hackeo, obtención de los gastos de personas y del mayor gastador, obtención de ID del puesto con menor stock para cierto ítem e información general del sistema, como las personas y los puestos que participan de él.

1.1. Interfaz

se explica con: LOLLAPATUZA.

géneros: `lolla`.

Operaciones básicas

NUEVOLOLLA(**in** *puestos* : `dicc(IdPuesto, puesto)`, **in** *personas* : `conj(personas)`) → *res* : `lolla`
Pre ≡ {claves(puestos) ≠ ∅ ∧ personas ≠ ∅ ∧ todosLosPuestosMismoPrecio(puestos) ∧ noVendieronAun(puestos)}
Post ≡ {res = crearLolla(puestos,personas)}
Complejidad: $O(A * log(A) + PAI^2)$
Descripción: inicia un nuevo sistema.
Aliasing: puestos y personas se guardan por copia

NUEVACOMPRA(**in/out** *l* : `lolla`, **in** *per* : `persona`, **in** *pi* : `puestoId`, **in** *i* : `item`, **in** *c* : `cant`)
Pre ≡ {l_0 = l ∧ per ∈ personas(l) ∧ def?(pi,puestos(l)) ∧$_L$haySuficiente?(obtener(pi,puestos(l)), i, c) }
Post ≡ {l = vender(l_0, pi, per, 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.

HACKEARITEM(**in/out** *l* : `lolla`, **in** *i* : `item`,**in** *per* : `persona`)
Pre ≡ {l_0 = l ∧ consumióSinPromoEnAlgunPuesto(l, per, i) }
Post ≡ {l = hackear(l_0, per, i)}
Complejidad: O(log(A)+log(I))/O(log(A)+log(I)+log(P))
Descripción: Hackea un ítem consumido por una persona. Se hackea el puesto de menor ID en el que la persona haya consumido ese ítem sin promoción.

GASTOPERSONA(**in** *l* : `lolla`, **in** *per* : `persona`) → *res* : `dinero`
Pre ≡ {per ∈ personas(l)}
Post ≡ {res = gastoTotal(l, per)}
Complejidad: O(log(A))
Descripción: Obtiene el gasto total de una persona.

MAYORCOMPRADOR(**in** *l* : `lolla`) → *res* : `persona`
Pre ≡ {true}
Post ≡ {res = masGasto(l)}
Complejidad: O(1)
Descripción: Obtiene la persona que más dinero gastó. Si hay más de una persona que gastó el monto máximo, desempata por ID de la persona.

MENORSTOCK(**in** *l* : `lolla`, **in** *i* : `item`) → *res* : `idPuesto`
Pre ≡ {true}
Post ≡ {obtener(res, puestos(l)) = menorStock(l, i)}
Complejidad: O(P ∗ log(I))
Descripción: Dado un ítem, devuelve el puesto que tenga menor stock. Si más de un puesto tiene el mínimo stock, devuelve el de menor ID

obtenerPersonas(**in** l: `lolla`) $\rightarrow$ res : `conj(persona)`
Pre $\equiv$ {true}
Post $\equiv$ {res = personas(l)}
Complejidad: O(1)
Descripción: Obtiene las personas del sistema.

obtenerPuestos(**in** l: `lolla`) $\rightarrow$ res : `diccLog(idPuesto, puesto)`
Pre $\equiv$ {true}
Post $\equiv$ {res = puestos(l)}
Complejidad: O(1)
Descripción: Obtiene los puestos de comida con sus IDs.

1.2. Representación

`lolla` se representa con `l`

donde `l` es `tupla`(
mayorGastoXpersonas: `DiccLog(personaYGasto, persona)`,
infoPersonas: `DiccLog(persona, tupla<gasto: gasto, it : itDiccLog>)`,
Puestos: `DiccLog(idPuesto, puesto)`,
puestosHackeables: `DiccLog(persona, DiccLog(item, diccLog(idPuesto, itDiccLog))`
)

▷ personaYGasto es tupla(persona, gasto)

▷ $(\forall\ t_1, t_2 : \text{personaYGasto})(t_1 < t_2 \iff (t_1.\text{gasto} > t_2.\text{gasto}) \vee (t_1.\text{gasto} = t_2.\text{gasto} \wedge t_1.\text{persona} < t_2.\text{persona}))$

Rep : lolla $\longrightarrow$ bool

Rep(l) $\equiv$ (1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4) $\wedge$ (5) $\wedge$ (6)

(1) $\equiv$ ($\forall$ per : persona)(per $\in$ claves(l.infoPersonas) $\Rightarrow$ (1a) $\wedge$ (1b) $\wedge$ (1c))

(1a) $\equiv$ π_1 (obtener(per, l.infoPersonas)) = $\sum_{id \in claves(l.Puestos)}$ gastoDe(obtener(id, l.Puestos), per)

(1b) $\equiv$ (π_1(Siguiente(π_2 (obtener(per, l.infoPersonas))))) = per

(1c) $\equiv$ π_2(Siguiente(π_2(obtener(per, l.infoPersonas)))) = π_1(obtener(per, l.infoPersonas))

(2) $\equiv$ ($\forall\ id_1, id_2$: idPuesto)((2a) $\Rightarrow$ (2b))

(2a) $\equiv$ ($id_1 \neq id_2 \wedge id_1 \in$ claves(l.Puestos) $\wedge\ id_2 \in$ claves(l.Puestos))

(2b) $\equiv$ obtener(id_1, l.Puestos) $\neq$ obtener(id_2, l.Puestos)

(3) $\equiv$ ($\forall\ t_1, t_2$: tupla(persona, gasto))
((t$_1$ $\in$ claves(l.mayorGastoXPersonas) $\wedge$ t$_2$ $\in$ claves(l.mayorGastoXPersonas)) $\Rightarrow$ π_1(t$_1$) $\neq$ π_1(t$_2$))

(4) $\equiv$ ($\forall$ t : tupla(persona, gasto))
(t $\in$ claves(l.mayorGastoXPersonas) $\Rightarrow$ (4a))

(4a) $\equiv$ π_1(t) $\in$ claves(l.infoPersonas) $\wedge$
π_2(t) = π_1(obtener(π_1(t), l.infoPersonas)) $\wedge$
obtener(t, l.mayorGastoXPersonas) = π_1(t)

(5) $\equiv$ ($\forall$p : persona)(p $\in$ claves(l.puestosHackeables) $\Rightarrow$ (5a))

(5a) $\equiv$ ($\forall$ i : item)(i $\in$ claves(obtener(per, l.puestosHackeables)) $\Rightarrow$ (5b)

(5b) $\equiv$ ($\forall$ id : idPuesto)(id $\in$ claves(obtener(i, obtener(per, l.puestosHackeables))) $\Rightarrow$ (5c))

(5c) $\equiv$ ($\exists$ t : tupla(persona, gasto))(π_1(t) = i $\wedge$ (5d) $\wedge$ (5e))

(5d) $\equiv$ t $\in$ ventas(SiguienteSignificado(obtener(id, obtener(i, obtener(per, l.puestosHackeables)))))

(5e) $\equiv$ descuento(SiguienteSignificado(obtener(id, obtener(i, obtener(per, l.puestosHackeables)))), i, π_2(t)) = 0

(6) $\equiv$ (claves(l.infoPersonas) = claves(l.puestosHackeables) $\wedge$
($\forall$ p : persona) (p $\in$ claves(l.infoPersonas) $\iff$ (6a))

(6a) $\equiv$ ($\exists$ t : tupla(persona, gasto))
(t $\in$ claves(l.mayorGastoXPersonas) $\wedge$ π_1(t) = p)

Abs : lolla l $\longrightarrow$ lolla {Rep(l)}

Abs(l) $\equiv$ lo : lolla / (puestos(lo) = l.Puestos $\wedge$ personas(lo) = claves(l.infoPersonas)

1.3. Algoritmos

NUEVOLOLLA(**in** *puestos* : `diccLog(idPuesto, puesto)`, **in** *personas* : `conj(persona)`) $\longrightarrow$ `res : lolla`

```
1:  itConj itP = CrearIt(personas)                                                          ▷ O(1)
2:  diccLog(tupla(persona, gasto), persona) mayorGastoXPersonas = Vacío()                   ▷ O(1)
3:  diccLog(persona, tupla(gasto, itDiccLog)) infoPersonas = Vacío()                        ▷ O(1)
4:  diccLog(persona, diccLog(item, diccLog(idPuesto, itDiccLog))) puestosHackeables = Vacío() ▷ O(1)
5:  mientras HaySiguiente(itP) hacer                                                        ▷ O(A * log (A))
6:      itDiccLog it = Definir(l.mayorGastoXpersonas, tupla(Siguiente(itP), 0), Siguiente(itP))  ▷ O(log (A))
7:      Definir(l.infoPersona, Siguiente(itP), tupla(0, it))                                ▷ O(log (A))
8:      Definir(l.puestosHackeables, Siguiente(itPersonas), Vacía())                        ▷ O(log(A))
9:      Avanzar(itPersonas)                                                                 ▷ O(1);
10: fin mientras
11: devolver tupla(mayorGastoXPersona, infoPersonas, puestos, puestosHackeables)            ▷ O(PAI^2)
```

▷ **Complejidad:** $O(A * log(A) + PAI^2)$

NUEVACOMPRA(**inout** l : `lolla`, **in** *per* : `persona`, **in** pi : `puestoId`, **in** i : `item`, **in** c : `cant`)

```
1:  stockItem(i, Significado(l.Puestos, pi)) = stockItem(i, Significado(l.Puestos, pi)) - c   ▷ O(log (P) + log (I))
2:  nat descuento = descuentoDe(i, c, Significado(l.Puestos, pi))                             ▷ O(log (P) + log (I) + log (Cant))
3:  si (descuento == 0) entonces
4:      si (Definido?(Significado(l.puestosHackeables, per), i) == false)                     ▷ O(log (A) + log (I)) entonces
5:          Definir(Significado(l.puestosHackeables, per), i, Vacío())                        ▷ O(log (A) + log (I))
6:      fin si
7:      Definir(
8:          Significado(Significado(l.puestosHackeables, per), i),
9:          id,
10:         IteradorA(l.Puestos, id)
11:     )                                                                                     ▷ O(log (A) + log (I) + log (P))
12:     AgregarAdelante(Significado(Ventas(Significado(l.Puestos, pi), per), i), c)           ▷ O(log (P) + log (A) + log (I))
13: sino
14:     AgregarAtras(Significado(Ventas(Significado(l.Puestos, pi), per), i), c)              ▷ O(log (P) + log (A) + log (I))
15: fin si
16: nat precio = precioItem(i, Significado(l.Puestos, pi))                                   ▷ O((log (P) + log (I)))
17: nat gasto = c * (precio*((100-descuento)/100))                                           ▷ O(1)
18: gastoPersona(Significado(l.Puestos, pi), per) = gastoPersona(Significado(l.Puestos, pi), per)
19:                                                 + gasto                                  ▷ O(log (P) + log (A))
20: nat gastoTotal = Significado(l.infoPersonas, per).gasto + gasto                          ▷ O(log (A))
21: EliminarSiguiente(Significado(l.infoPersonas, per).itDiccLog)                            ▷ O(log (A))
22: itDiccLog it = Definir(l.mayorGastoXPersonas, tupla(per, gastoTotal, per)               ▷ O(log (A))
23: Definir(l.infoPersonas, per, tupla(gastoTotal, it))                                      ▷ O(log (A))
```

▷ **Complejidad: O(log (P) + log (I) + log (A) + log (Cant))**

HACKEARITEM(**inout** l : `lolla`, **in** i : `item`, **in** p : `persona`)

```
1:  itDiccLog hackeando1 = crearIt(Significado(Significado(l.puestosHackeables, p), i)        ▷ O(log(A)+log(I))
2:  itDiccLog hackeando = SiguienteSignificado(hackeando1)                                    ▷ O(1)
3:  stockItem(SiguienteSignificado(hackeando), i)++                                           ▷ O(log(I))
4:  Primero(Ventas(SiguienteSignificado(hackeando), per))−−                                   ▷ O(log(A))
5:  si Primero(Ventas(SiguienteSignificado(hackeando), p)) == 0 entonces                      ▷ O(log(A))
6:      Fin(Ventas(SiguienteSignificado(hackeando), per))                                     ▷ O(1)
7:  fin si
8:  si
9:      descuentoDe(SiguienteSignificado(hackeando), i, Primero(Ventas(SiguienteSignificado(hackeando), p)) ≠ 0
10:  entonces                                                                                 ▷ O(log(I)+log(A))
11:     EliminarSiguiente(hackeando1)                                                         ▷ O(1)
12: fin si
13: nat gastoActualizado = Significado(l.infoPersonas, p).gasto
14:                          - precioItem(SiguienteSignificado(hackeando), i)                 ▷ O(log(I)+log (A))
15: EliminarSiguiente(Significado(l.infoPersonas, p).it)                                      ▷ O(log(A))
16: itDiccLog it = Definir(l.mayorGastoXPersonas, tupla(p, gastoActualizado), p)              ▷ O(log(A))
17: Definir(l.infoPersonas, p, tupla(gastoActualizado, it))                                   ▷ O(log(A))
```

▷ **Complejidad: O(log(A)+log(I))**

OBTENERPERSONAS(**in** l : `lolla`) $\longrightarrow$ `res : lista(persona)`

```
1: devolver Claves(l.infoPersonas)                                                            ▷ O(1)
```

▷ **Complejidad: O(1)**

OBTENERPUESTOS(**in** l : `lolla`) $\longrightarrow$ `res : diccLog(idPuesto, puesto)`

```
1: devolver l.Puestos                                                                         ▷ O(1)
```

▷ **Complejidad: O(1)**

menorStock(**in** l : `lolla`, **in** i : `item`) $\longrightarrow$ `res : idPuesto`

```
1:  itDiccLog itPuestos = CrearIt(l.Puestos)                                                          ▷ O(1)
2:  tupla(idPuesto, cant) candidato =
3:                              (SiguienteClave(itPuestos), stockItem(i, SiguienteSignificado(itPuestos)))   ▷ O(log I)
4:  int contadorSinItemEnMenu = 0                                                                     ▷ O(1)
5:  idPuesto candidatoPorSiNingunoTieneItemEnMenu = SiguienteClave(itPuestos)                         ▷ O(1)
6:  mientras HaySiguiente(itPuestos) hacer                                                            ▷ O(P * log i)
7:      si enMenú?(SiguienteSignificado(itPuestos), i) entonces                                       ▷ O(log I)
8:          si stockItem(i, SiguienteSignificado(itPuestos)) < candidato.second entonces              ▷ O(log I)
9:              candidato = (SiguienteClave(itPuestos), stockItem(i, SiguienteSignificado(itPuestos)))  ▷ O(log I)
10:         sino
11:             si
12:                 stockItem(i, SiguienteSignificado(itPuestos)) == candidato.second) &&
13:                 SiguienteClave(itPuestos) < candidato.first
14:             entonces                                                                              ▷ O(log I)
15:                 candidato = (SiguienteClave(itPuestos), candidato.second)                         ▷ O(1)
16:             fin si
17:         fin si
18:     sino
19:         contadorSinItemEnMenu++                                                                   ▷ O(1)
20:         si SiguienteClave(itPuestos) < candidatoPorSiNingunoTieneItemEnMenu entonces              ▷ O(1)
21:             candidatoPorSiNingunoTieneItemEnMenu = SiguienteClave(itPuestos)                      ▷ O(1)
22:         fin si
23:     fin si
24:     Avanzar(itPuestos)                                                                            ▷ O(1)
25: fin mientras
26: si contadorSinItemEnMenu == #Claves(l.Puestos) entonces                                           ▷ O(1)
27:     devolver candidatoPorSiNingunoTieneItemEnMenu                                                 ▷ O(1)
28: fin si
29: devolver candidato.first                                                                          ▷ O(1)
```

▷ **Complejidad: P * log I**

gastoPersona(**in** l : `lolla`, **in** per : `persona`) $\longrightarrow$ `res : dinero`

```
1:  devolver Significado(l.gastoXPersona, per).gasto                                                  ▷ O(log A)
```

▷ **Complejidad: O(log A)**

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

```
1:  itDiccLog it = CrearIt(l.mayorGastoXPersonas)                                                     ▷ O(1)
2:  devolver SiguienteSignificado(it)                                                                 ▷ O(1)
```

▷ **Complejidad: O(1)**

2. Módulo Puesto de Comida

El módulo Puesto de Comida provee una estructura que permite administrar la inicialización de un puesto, conocer el stock de un cierto ítem y sus descuentos disponibles por cantidad adquirida y obtener el gasto total de una persona en el puesto.

2.1. Interfaz

se explica con: PUESTODECOMIDA.

géneros: `puesto`.

Operaciones básicas

NUEVOPUESTO(**in** *stock*: `dicc(item, nat)`, **in** *precios*: `dicc(item, nat)`,**in** *descs*: `dicc(item, dicc(cant, nat=))` → *res* : `puesto`
Pre ≡ {claves(stock) = claves(precios) ∧ claves(descs) ⊆ claves(stock)}
Post ≡ {res = crearPuesto(precios, stock, descs)}
Complejidad: O(I^3 + (I^3)*log(I))
Descripción: inicializa un nuevo puesto.
Aliasing: stock, precios y descs son añadidos por copia al nuevo puesto

STOCKÍTEM(**in** *i*: `item`, **in** *p*: `puesto`) → *res* : `cant`
Pre ≡ {i ∈ menu(p)}
Post ≡ {res = stock(p,i)}
Complejidad: O(log(I))
Descripción: obtiene el stock de un ítem.

PRECIOÍTEM(**in** *i*: `item`,**in** *p*: `puesto`) → *res* : `dinero`
Pre ≡ {i ∈ menu(p)}
Post ≡ {res = precio(p, i)}
Complejidad: O(log(I))
Descripción: devuelve el precio de un item.

DESCUENTODE(**in** *i*: `item`,**in** *c*: `cant`,**in** *p*: `puesto`) → *res* : `nat`
Pre ≡ {i ∈ menu(p)}
Post ≡ {res = descuento(p,i,c)}
Complejidad: O(log(I)+log(cant))
Descripción: obtiene el descuento de un ítem dada la cantidad del mismo.

GASTOPERSONA(**in** *per*: `persona`, **in** *p*: `puesto`) → *res* : `nat`
Pre ≡ {true}
Post ≡ {res = gastoDeVentas(p, ventas(p,per))}
Complejidad: O(log(A))
Descripción: obtiene el gasto realizado por una persona en el puesto.

VENTAS(**in** p: `puesto`,**in** per: `persona`) → res : `diccLog(item, lista(cant))`
Pre ≡ {true}
Post ≡ {
(∀ i : item)(
i ∈ claves(res) ⇒ (
(∀ t : tupla(item, cant))(
i=π_1(t) ⇒
#Apariciones(t, ventas(p, per)) = #Apariciones(π_2(t),obtener(π_1(t), res))
)
)
)
}
Complejidad: O(log (A))
Descripción: Dada una persona, devuelve un diccionario con una lista de cantidades compradas por item.
Aliasing: res es una referencia modificable

COPIAR(**in** p: `puesto`) → res : `puesto`
Pre ≡ {true}
Post ≡ {res =$_{obs}$ p}
Complejidad: $O(AI^2)$
Descripción: Devuelve un nuevo puesto

2.2. Representación

`puesto` se representa con `p`

donde `p` es `tupla`(
stock: `diccLog(item, cant)`,
precios: `diccLog(item, precio)`,
descuentos: `diccLog(item, diccLog(cant, descuento))`,
ventas: `diccLog(persona, diccLog(item, lista(cant)))`,
gastoPersona: `diccLog(persona, cant)`
)

con diccDescuentos = diccLog(item, tupla(maxCant : cant, descuentos : diccLog(cant, descuento)))

Rep : puesto $\longrightarrow$ bool

Rep(p) $\equiv$ (1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4) $\wedge$ (5) $\wedge$ (6)

(1) $\equiv$ claves(p.stock) = claves(p.precios) $\wedge$
claves(p.descuentos) $\subseteq$ claves(p.stock)$\wedge$
claves(p.ventas) = claves(p.gastoPersona)$\wedge$
($\forall$ per : persona)
(per $\in$ claves(p.ventas) $\Rightarrow$ claves(obtener(per, p.ventas)) = claves(p.stock)))

(2) $\equiv$ ($\forall$ i : item)
(i $\in$ claves(p.precios) $\Rightarrow$ obtener(i, p.precios) > 0)

(3) $\equiv$ ($\forall$ per : persona)
(per $\in$ claves(p.gastoPersona) $\Rightarrow$ (3a))

(3a) $\equiv$ obtener(per, p.gastoPersona) = calcularGastoPersona(obtener(per, p.ventas), p.precios, p.descuentos)

(4) $\equiv$ ($\forall$ i : item)
(i $\in$ claves(p.descuentos) $\Rightarrow$ ((4a) $\wedge$ (4b))

(4a) $\equiv$ π_2(obtener(i, p.descuentos)) $\neq$ vacía

(4b) $\equiv$ ($\forall$ c : cant)
(c $\in$ claves((π_2(obtener(i, p.descuentos))) $\Rightarrow$ c > 0 $\wedge$ 0 < obtener(c, (π_2(obtener(i, p.descuentos)))<100))

(5) $\equiv$ ($\forall$ per : persona)
(per $\in$ claves(p.ventas) $\Rightarrow$ (5a))

(5a) $\equiv$ ($\forall$ i : item)
(i $\in$ claves(obtener(per, p.ventas)) $\Rightarrow$ (5b))

(5b) $\equiv$ ($\forall$ c : cant)
(c $\in$ obtener(i, obtener(per, p.ventas)) $\Rightarrow$ c>0)

(6) $\equiv$ ($\forall$ i : item)
(i $\in$ claves(p.descuentos) $\Rightarrow$ (6a) $\wedge$ (6b) $\wedge$ (6c))

(6a) $\equiv$ π_1(obtener(i, p.descuentos)) $\in$ claves(π_2(obtener(i, p.descuentos)))

(6b) $\equiv$ $\neg$($\exists$ c : cant)(c $\in$ claves(π_2(obtener(i, p.descuentos)))

(6c) $\equiv$ c > π_1(obtener(i, p.descuentos))

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

Abs(p) $\equiv$ pues : puesto /
(menu(pues) = claves(p.precios) $\wedge$

($\forall$ i : item)(i $\in$ menu(pues) $\Rightarrow$ precio(pues, i) = obtener(i, p.precios)) $\wedge$

($\forall$ i : item)(i $\in$ menu(pues) $\Rightarrow$ stock(pues, i) = obtener(i, p.stock)) $\wedge$

($\forall$ i : item)($\forall$ c : cant)
(i $\in$ menu(pues) $\Rightarrow$ descuento(pues, i, c) = mejorDescuento(obtener(i, p.descuentos), c)) $\wedge$

($\forall$ per : persona)($\forall$ v :multiconj(item, cant))
(#(v, ventas(pues, per)) = #Apariciones(π_2(v), obtener(π_1(v), obtener(per, p.ventas)))))

2.3. Algoritmos

NUEVOPUESTO(**in** *stock* : `diccLog(item, nat)`, **in** *precios* : `diccLog(item, nat)`,**in** *descs* : `diccLog(item, dicc(cant, nat)))` $\longrightarrow$ `res : puesto`

```
1:  itDiccLog itItemDesc = CrearIt(descs)                                          ▷ O(1)
2:  diccDescuentos descuentos = Vacío()                                            ▷ O(1)
3:  mientras HaySiguiente(itItemDesc) hacer                                        ▷ O(1)
4:      Definir(descuentos, SiguienteClave(itItemDesc), tupla(0, Vacío()))         ▷ O(1)
5:      itDiccLog itCantItem = CrearIt(Significado(descs, SiguienteClave(itItemDesc)).second)   ▷ O(1)
6:      nat maxCant;
7:      nat minCant;
8:      rellenarDescXCant(itItemDesc, itCantItem, descuentos, maxCant, minCant)    ▷ O(I^2 + (I^2)*log(I))
9:      Definir(descuentos,                                                        ▷ O(1)
10:             SiguienteClave(itItemDesc),                                        ▷ O(1)
11:             tupla(maxCant, minCant, Significado(descuentos, SiguienteClave(itItemDesc).second)))   ▷ O(log(I))
12:     Avanzar(itItemDesc)                                                        ▷ O(1)
13: fin mientras
14: diccLog(persona, diccLog(item, lista(cant))) ventas = Vacía()                  ▷ O(1)
15: diccLog(persona, nat) gastoPersona = Vacía()                                   ▷ O(1)
16: devolver tupla(stock, precios, descuentos, ventas, gastoPersona)
```

▷ **Complejidad: O(I^3 + (I^3)*log(I))**

RELLENARDESCXCANT(**in** *itItemDesc* : `itDiccLog`, **in** *itCantItem* : `itDiccLog`, **in** *descuentos* : `diccDescuentos`, **out** *maxCant* : `nat`, **out** *minCant* : `nat`)

```
1:  maxCant = SiguienteClave(itItemDesc)                                           ▷ O(1)
2:  minCant = SiguienteClave(itItemDesc)                                           ▷ O(1)
3:  mientras HaySiguiente(itCantItem) hacer
4:      Definir(Significado(descuentos, SiguienteClave(itItemDesc)).descuentos,    ▷ O(log(I))
5:              SiguienteClave(itCantItem),                                        ▷ O(1)
6:              SiguienteSignificado(itCantItem))                                  ▷ O(1)
7:                                                                                 ▷ O(1+log(I))=O(log(I))
8:      si itCantItem.clave > maxCant entonces
9:          maxCant = SiguienteClave(itCantItem)                                   ▷ O(1)
10:         si itCantItem.clave < minCant entonces
11:             minCant = SiguienteClave(itCantItem)                               ▷ O(1)
12:         fin si
13:         rellenarEntreDescuentos(itItemDesc, itCantItem)                        ▷ O(I + I*log(I))
14:         Avanzar(itCantItem)                                                    ▷ O(1)
15:     fin si
16: fin mientras
17:                                                                                ▷ O(I^2 + I^2*log(I))
```

rellenarEntreDescuentos(**in** *itItemDesc* : `itDiccLog`, **in** *itCantItem* : `itDiccLog` : `nat`)

```
1:  int cantidad = SiguienteClave(itCantItem) - 1
2:  minCant = SiguienteClave(itItemDesc)                                          ▷ O(1)
3:  //guarda: O(log(mCDI)); interior: O(1); iteraciones < mCDI
4:  //total ciclo: O(mCID*log(mCDI)) = O(I*log(I)
5:  mientras cantidad > 0 || definido?(cantidad, Siguiente(itItemDesc).second) == false hacer
6:      cantidad = cantidad - 1
7:  fin mientras
8:  descuento = si (cantidad == 0) entonces 0
9:                  sino Significado(Siguiente(itItemDesc).second,cantidad)       ▷ O(log(mCDI)) = O(log(I))
10: //guarda: O(1); interior: O(1); iteraciones: cantActual - cantAnterior = k < mCID
11: //total ciclo: O(1*k)=O(k)=O(mCID)=O(I)
12: mientras cantidad < SiguienteClave(itCantItem) hacer
13:     Definir(Siguiente(itItemDesc).second,                                     ▷ O(1)
14:             cantidad,
15:             descuento)                                  ▷ O(copy(cantidad)+copy(descuento)) = O(1)
16:     cantidad = cantidad + 1                                                   ▷ O(1)
17: fin mientras
18:                          ▷ O(mCID*log(mCDI)+log(mCDI)+mCID) = O(I+log(I)+I*log(I)) = O(I+I*log(I))
```

Donde mCDI es la mayor cantidad de descuentos que puede tener un item < mCID, y mCID es la mayor cantidad de items para la que puede haber descuento < I

enMenú?(**in** *pues* : `puesto`, **in** *i* : `item`) ⟶ `res : bool`

```
1: devolver Definido?(i, pues.stock)                                              ▷ O(log(I))
```

▷ **Complejidad: O(log(I))**

stockItem(**in** *pues* : `puesto`, **in** *i* : `item`) ⟶ `res : cant`

```
1: int res = significado(i, pues.stock)                                           ▷ O(log(I))
2: devolver res
```

▷ **Complejidad: O(log(I))**

precioItem(**in** *pues* : `puesto`, **in** *i* : `item`) ⟶ `res : cant`

```
1: int res = significado(i, pues.precio)                                          ▷ O(log(I))
2: devolver res
```

▷ **Complejidad: O(log(I))**

maxDescuentoItem(**in** *pues* : `puesto`, **in** *i* : `item`) ⟶ `res : cant`

```
1: devolver Significado(pues.descuentos, i).maxCant                               ▷ O(log(I))
```

▷ **Complejidad: O(log(I))**

descuentoDe(**in** i : `item`, **in** c : `cant`, **in** p : `puesto`) $\longrightarrow$ `res : nat`

```
1:
2: si c <Significado(p.descuentos, i).minCant entonces
3:     devolver 0                                                    ▷ O(log(I))
4: fin si
5:
6: si c >maxDescuentoItem(i, p) entonces                             ▷ O(log(I))
7:     devolver Significado( Significado(p.descuentos, i).second, maxDescuentoItem(i, p))
8:                                                                   ▷ O(2*log(I) + log(cant))
9: fin si
10: devolver Significado(Significado(p.descuentos, i).second, c)     ▷ O(log(I) + log(cant))
```

▷ **Complejidad: O(log(I) + log(cant))**

gastoPersona(**in** p : `puesto`, **in** a : `persona`) $\longrightarrow$ `res : cant`

```
1: int res = significado(p.gastoPersona, a)                          ▷ O(log(A))
       devolver res
```

▷ **Complejidad: O(log(A))**

Ventas(**in** p : `puesto`, **in** a : `persona`) $\longrightarrow$ `res : diccLog(item, lista(cant))`

```
1: devolver Significado(p.Ventas, a)
```

▷ **Complejidad: O(log(A))**