Algoritmos y Estructuras de Datos II

Trabajo Práctico 2

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

Lollapatuza

Integrante	LU	Correo electrónico
Agustin Fernandez Aragon	998/21	f.a.agustin@gmail.com
Bruno Muschietti	924/21	brunomuschi@gmail.com
Felipe Saidón	1436/21	felipesaidon@gmail.com
Matias Daniel Diaz Sarmiento	704/19	mdds.2017@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Modulo Puesto de Comida

Interfaz

se explica con: PuestoDeComida

géneros: `puesto`.

1.1. Operaciones básicas de Puesto de Comida

Nuevo(**in** *menu* : `dicc(item, nat)`, **in** *stock* : `dicc(item,nat)`, **in** *descuentos* : `dicc(item, dicc(cant,nat) )` $\rightarrow$ *res* : `Puesto`
Pre $\equiv$ {$\neg$ Vacio?(claves(menu)) $\wedge_L$ ($\forall$i:item) def?(i,menu) $\rightarrow_L$ obtener(i,menu) > 0 $\wedge$ claves(stock) = claves(menu) $\wedge$ ($\forall$i:item) def?(i,stock) $\rightarrow_L$ obtener(i,stock) > 0 $\wedge$ ($\forall$i:item) def?(i,descuentos) $\rightarrow_L$ def?(i,menu) $\wedge$ ($\forall$i:item) def?(i,descuentos)$\rightarrow_L$($\forall$c:cant) def?(c,obtener(i,descuentos)) $\rightarrow_L$c > 0 $\wedge$ obtener(c,obtener(i,descuentos)) > 0 $\wedge$ obtener(c,obtener(i,descuentos)) < 100 }
Post $\equiv$ {res $=_{\text{obs}}$ crearPuesto(menu,stock,descuentos)}
Complejidad: $O(1)$
Descripción: Genera un puesto nuevo.
Aliasing: No aplica.

ObtenerStock(**in** *p* : `puesto`, **in** *i* : `item`) $\rightarrow$ *res* : `nat`
Pre $\equiv$ {i $\in$ menu(p)}
Post $\equiv$ {res $=_{\text{obs}}$ stock(p,i) }
Complejidad: $O(log(I))$
Descripción: Devuelve el stock de un item en un puesto dado
Aliasing: Se devuelve una referencia no modificable del stock de un item para un determinado puesto.

ObtenerDescuento(**in** *p* : `puesto`, **in** *i* : `item`, **in** *c* : `cant`) $\rightarrow$ *res* : `nat`
Pre $\equiv$ {i $\in$ menu(p)}
Post $\equiv$ {res $=_{\text{obs}}$ descuento(p,i,c)}
Complejidad: $O(log(I) + log(cant))$
Descripción: Devuelve el descuento de un item para una cierta cantidad
Aliasing: Se devuelve una referencia no modificable del valor del descuento para un determinado item y cantidad.

ObtenerGasto(**in** *p* : `puesto`,**in** *per* : `persona`) $\rightarrow$ *res* : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{\text{obs}}$ gastosDe(p,per)}
Complejidad: $O(log(A))$
Descripción: Devuelve el gasto de una persona en un puesto
Aliasing: Se devuelve un referencia no modificable al significado del diccionario p.gastoPorPersona para la persona a.

Venta(**in/out** *p* : `puesto`, **in** *a* : `persona`, **in** *i* : `item`, **in** *c* : `cant`)
Pre $\equiv$ {p $=_{\text{obs}}$ p_0 $\wedge$ i $\in$ menu(p) $\wedge_L$ c $\leq$ $stock(p, i)$}
Post$\equiv$ {p = vender(p_0,a,i,c)}
Complejidad: $O(log(I) + log(A) + log(cant))$
Descripción: Registra la venta realizada por una persona de cierto item y cantidad
Aliasing: Devuelve referencia al puesto modificado

HackearP(**in/out** *p* : `puesto`, **in** *a* : `persona`, **in** *i* : `item`)
Pre $\equiv$ {p $=_{\text{obs}}$ p_0}
Post $\equiv$ {p = olvidarItem(p,a,i) }
Complejidad: $O(log(I) + log(A))$
Descripción: Registra la venta realizada por una persona de cierto item y cantidad
Aliasing: Devuelve referencia al puesto modificado

Representación

1.2. Representación del Puesto de Comida

`puesto` **se representa con** `estr`

donde `estr` es `tupla`(*menu*: `diccLog(item,precio)`
, *stock*: `diccLog(item,cant)`
, *descuentos*: `diccLog(item,diccLog(cant,nat))`
, *gastoPorPersona*: `diccLog(persona,nat)`
, *ventas*: `diccLog(persona,diccLog(item, Lista(cant)))`
, *itVentasSinPromo*: `diccLog(persona,diccLog(item, lista(iteradorLISTA)))`
)

1) las claves de menu no son vacias

2) los precios de menu son mayores a 0

3) las cantidades de stock son mayores o iguales a 0

4) las claves de stock son iguales a las de menu

5) En descuentos, si un item no es vacio (es decir, está definido), entonces las claves del significado no son vacias

6) En descuentos, las claves del significado son mayores a 0 y todo significado de las claves del significado esta entre 1 y 99

7)claves de descuentos esta contenida en claves de menu

8)Para toda persona, el gasto por persona es mayor a 0 en gastoPorPersona

9)si persona definida en ventas, su significado no tiene ser vacio, ni la lista (significado del significado)

10) las cantidades de la lista de ventas tienen que ser mayores a 0

11)significado de itVentas no sea vacio (idem 9)

12) claves gastoPersona = claves ventas

13) claves itVentas contenido en claves ventas

14) No hay descuento menor a la cantidad marcada por el iterador en descuentos para ese item.

15) claves de significado en ventas y itVentas estan definidas en menu

16) para 1 persona definida en ventas y itVentas, los items de itVentas estan incluidos en el diccionario de items de ventas de esa persona

17) para una misma persona, la longitud de su lista de iteradores de ventas sin promo es menor igual a la longitud de la lista de ventas

18) todos los iteradores en la lista de iteradores en itVentasSinPromo apuntan a una cantidad comprada del item en una compra que se hizo sin promoción

Rep : est $\longrightarrow$ bool

Rep(e) $\equiv$ true $\Longleftrightarrow$

1) $\neg$ vacío?(claves(e.menu))

2) ($\forall$i: item)(def?(i,e.menu) $\rightarrow_L$ obtener(i,e.menu)>0)

3)($\forall$i: item)(def?(i,e.stock) $\rightarrow$ obtener(i,e.stock)$\geq$0)

4)claves(e.menu)$=_{\text{obs}}$ claves(e.stock)

5)($\forall$i: item)(def?(i,e.descuentos) $\rightarrow$ $\neg$ vacío?(claves(obtener(i,e.descuentos))))

6)($\forall$i: item)(def?(i,e.descuentos)$\rightarrow_L$ sonPositivos?(claves(obtener(i,e.descuentos)))
$\wedge$ susSignEstánEnRango(obtener(i,e.descuentos)))

7)claves(e.descuentos)$\subset$claves(e.menu)

8)($\forall$p: persona)(def?(p,e.gastoPorPersona)$\rightarrow$ obtener(p,e.gastoPorPersona)>0)

9)($\forall$p: persona)(def?(p,e.ventas)$\rightarrow_L$ $\neg$ vacío?(claves(obtener(p,e.ventas)))
$\wedge$ NotieneSignVacíos?(obtener(p,e.ventas)))

10)($\forall$p: persona)(def?(p,e.ventas)$\rightarrow_L$ susCantSonPositivas?(obtener(p,e.ventas)))

11)($\forall$p: persona)(def?(p,e.itVentasSinPromo)$\rightarrow_L$ $\neg$ vacío?(claves(obtener(p,e.itVentasSinPromo)))
$\wedge$ NotieneSignVacíos?(obtener(p,e.itVentasSinPromo)))

12)claves(e.gastoPorPersona)$=_{\text{obs}}$ claves(e.itVentasSinPromo)

13)claves(e.itVentas)$\subset$ claves(e.ventas)

14)($\forall$p: persona) def?(p,e.itVentasSinPromo) $\rightarrow_L$($\forall$i: item)def?(i,obtener(p,e.itVentasSinPromo))
$\rightarrow_L$ noHayDescuento(i, actual(obtener(i, obtener(p,e.itVentasSinPromo))), e.descuentos)

15)($\forall$p: persona) def?(p,e.itVentasSinPromo) $\rightarrow_L$($\forall$i: item)def?(i,obtener(p,e.itVentasSinPromo)) $\rightarrow_L$ def?(i,e.menu)

16)($\forall$p: persona) def?(p,e.ventas) $\rightarrow_L$($\forall$i: item)def?(i,obtener(p,e.ventas)) $\rightarrow_L$ def?(i,e.menu)

17)($\forall$p: persona) def?(p,e.ventas) $\rightarrow_L$($\forall$i: item)def?(i,obtener(p,e.ventas))
$\rightarrow_L$ $|obtener(obtener(e.itVentasSinPromo, a), i)| \leq |obtener(obtener(e.ventas, a), i)|$

18)($\forall$p: persona)def?(p,e.itventasSinPromo) $\rightarrow_L$($\forall$i: item)def?(i,obtener(p,e.itVentasSinPromo)) $\rightarrow_L$ ($\forall$it: iterador) it $\in$ obtener(i,obtener(p,e.itVentasSinPromo)) $\rightarrow_L$ anteriores(it) $\circ$ siguientes(it) = secuDeCantSinPromo(p,i,obtener(i,obtener(p,e.ventas)))

1.2.1. Funciones Auxiliares del Rep de puestos de comida

sonPositivos?(conj)=($\forall$c: cant)c $\in$ conj $\rightarrow$ c $>$ 0

susSignEstanEnRango(dicc)=($\forall$c : cant)def?(c,dicc) $\rightarrow$ obtener(c,dicc) $>$ 0 $\wedge$ obtener(c,dicc) $<$ 100

NotieneSignVacios(dicc)=($\forall$i : item)def?(i,dicc) $\rightarrow$ $\neg$ vacia(obtener(i,dicc))

susCantSonPositivas?(dicc)=($\forall$i : item)def?(i,dicc) $\rightarrow$ ($\forall$c : cant) esta?(c,obtener(i,dicc)) $\rightarrow$ c $>$ 0

noHayDescuento(item,cant,dicc)=($\forall$c : cant)c $\in$ claves(obtener(item,dicc)))$\rightarrow$ cant $<$ c

secuDeCantSinPromo(persona, item,obtener(i,obtener(p,e.ventas)))=

if vacia?(obtener(i,obtener(p,e.ventas)))
then <> **else if** prim(obtener(i,obtener(p,e.ventas))) > min(claves(obtener(i,descuentos))
then secuDeCantSinPromo(persona,item,fin(obtener(i,obtener(p,e.ventas)))
else obtener(i,obtener(p,e.ventas)) • secuDeCantSinPromo(persona, item, fin(obtener(i,obtener(p,e.ventas))))

1.2.2. Abs de Puestos de Comida

Abs : estr e $\longrightarrow$ puesto {Rep(e)}
Abs(e) $\equiv$ (∀e: estr)Abs(e) $=_{obs}$ p :puesto /
claves(e.menu) $=_{obs}$ menu(p) ∧
(∀i: item)def?(i,e.menu) $\rightarrow_L$ obtener(i,e.menu) = precio(p,i) ∧
(∀i: item)def?(i,e.stock) $\rightarrow_L$ obtener(i,e.stock) = stock(p,i) ∧
(∀i: item)def?(i,e.descuentos) $\rightarrow_L$ (∀c:cant)def?(c, obtener(i,e.descuentos))$\rightarrow_L$
obtener(c,obtener(i,e.descuentos)) = descuento(p,i,c) ∧
(∀a: persona)def?(i,e.ventas) $\rightarrow_L$ (∀i:item, c: cant)def?(i, obtener(a, e.ventas)) $\rightarrow_L$
#apariciones(c,obtener(i,obtener(a,e.ventas))) = #apariciones(<i,c>,multiconjASecu(ventas(p,a)))

1.2.3. Funciones Auxiliares del Abs

#apariciones(e,secu)=
if prim(secu) == e **then**
1 + #apariciones(e,fin(secu))
else
#apariciones(e,fin(secu))
end if

multiconjASecu(multiconj)=
if ∅?($multiconj$) **then**
<>
else
dameUno(multiconj) • multiconjASecu(sinUno(multiconj))
end if

Algoritmos

---

iNuevo(**in** *menu* : `dicc(item, nat)`, **in** *stock* : `dicc(item,nat)`, **in** *descuentos* : `dicc(item, dicc(cant,nat))` → *res* : Puesto

1: $res.menu \leftarrow$ menu
2: $res.stock \leftarrow$ stock
3: $res.descuentos \leftarrow$ descuentos
4: $res.gastoPorPersona \leftarrow$ dicc::Vacío()
5: $res.ventas \leftarrow$ dicc::Vacío()
6: $res.itVentasSinPromo \leftarrow$ dicc::Vacío()

Complejidad: $O(1)$ ▷

---

iObtenerStock(**in** p : `puesto`, **in** i : `item`) → *res* : *nat*

res ←significado(p.stock,i)

Complejidad: $O(log(I))$
Justificacion: obtener significado de diccLog tiene complejidad logaritmica de la cantidad de claves definidas

---

---

iObtenerDescuento(**in** p : `puesto`, **in** i : `item`, **in** c : `cant`) → *res* : *nat*

if definido?(p.descuento, i) **then O(log(I))**
  if definido?(significado(p.descuentos, i), c) **then O(log(cant) + log(I))**
    res ←significado(significado(p.descuento, i), c) **O(log(cant) + log(I))**
  else
    it ← definir(significado(p.descuentos, i), c, 1) **O(log(cant) + log(I))**
    if HayAnterior?(it) **then**O(1)
      res ← AnteriorSignificado(it) **O(1)**
    else
      res ← 0 **O(1)**
    end if
  end if
  borrar(significado(p.descuentos, i), c) **O(log(cant) + log(I))**
else
  res ← 0 **O(1)**
end if

Complejidad: $O(log(I) + log(cant))$
Justificacion: Buscar el item tiene complejidad log(I), luego buscar la cantidad para ese item toma log(c) y todas las operiaciones del iterador toman O(1)

---

iObtenerGasto(**in** p : `puesto`, **in** a : `persona`) → *res* : *nat*

res ←significado(p.gastoPorPersona, a)

Complejidad: $O(log(A))$
Justificacion: Buscar la persona en el diccLog tiene complejidad log(A) con A la cantidad de personas)

---

iVentas(**in/out** p: `puesto`, **in** a: `persona`, **in** i: `item`, **in** c: `cant`)

1: $significado(p.stock, i)$ ← significado(p.stock, i) − cant **O(log(I))**
2: $significado(p.gastoPorPersona, a)$ ← significado(p.gastoPorPersona, a) + ((significado(p.menu, i) × c) − ((significado(p.menu, i)× c) × ObtenerDescuento(p,i,c))) **O(log(A) + log(I))**
3: **if** definido?(p.ventas, a) **then O(log(A))**
4: **if** definido?(significado(p.ventas, a), i) **then O(log(A) + log(I))**
5: **if** ObtenerDescuento(p,i,c) == 0 **then O(log(cant) + log(I))**
6: it ← AgregarAtras(significado(significado(p.ventas, a), i), c) **O(1 + log(A) + log(I))**
7: **if** definido?(p.itVentasSinPromo, a) **then O(log(A))**
8: definir(significado(p.itVentasSinPromo,a), i, it) **O(log(A) + log(I))**
9: **else**
10: definir(p.itVentasSinPromo, a, definir(vacio(), i, it)) **O(log(I))**
11: **end if**
12: **else**
13: AgregarAtras(significado(significado(p.ventas, a), i), c) **O(1 + log(A) + log(I))**
14: **end if**
15: **else**
16: definir(significado(p.ventas, a), i, vacia()) **O(log(A) + log(I))**
17: **if** ObtenerDescuento(p,i,c) == 0 **thenO(log(cant) + log(I))**
18: it ← AgregarAtras(significado(significado(p.ventas, a), i), c) **O(1 + log(A) + log(I))**
19: **if** definido?(p.itVentasSinPromo, a) **then O(log(A))**
20: definir(significado(p.itVentasSinPromo,a), i, it)**O(log(A) + log(I))**
21: **else**
22: definir(p.itVentasSinPromo, a, definir(vacio(), i, it)) **O(log(A)**
23: **end if**
24: **else**
25: AgregarAtras(significado(significado(p.ventas, a), i), c) **O(1 + log(A) + log(I))**
26: **end if**
27: **end if**
28: **else**
29: definir(p.ventas, a, definir(vacio(), i, Vacia())**O(log(A)**
30: **if** ObtenerDescuento(p,i,c) == 0 **thenO(log(cant) + log(I))**
31: it ← AgregarAtras(significado(significado(p.ventas, a), i), c)**O(1 + log(A) + log(I))**
32: definir(p.itVentasSinPromo, a, definir(vacio(), i, it))**log(A)**
33: **else**
34: AgregarAtras(significado(significado(p.ventas, a), i), c)**O(1 + log(A) + log(I))**
35: **end if**
36: **end if**

Complejidad: $O(log(I) + log(A) + log(cant))$
Justificacion: Son todas operaciones con complejidad log(A), log(I) ó log(cant), independientes entre sí.

iHackearP(**in/out** p : `puesto`, **in** a : `persona`, **in** i : `item`)

1: c ← primero(significado(significado(e.itVentasSinPromo, a), i))**O(log(A)+log(I)**
2: **if** c>1 **thenO(1)**
3: $siguiente(c)$ ← (siguiente(c) − 1) **O(1)**
4: **else**
5: elminarSiguiente(c) **O(1)**
6: **end if**
7: **if** esVacia?(significado(significado(e.ventas, a), i)) **then O(log(A)+log(I)**
8: borrar(significado(e.ventas, a), i) **O(log(A)+log(I))**
9: **if** #claves(e.ventas) == 0 **then O(1)**
10: borrar(e.ventas, a) **O(log(A)**
11: **end if**
12: **end if**
13: **if** esVacia?(fin(significado(significado(itVentasSinPromo, a), i)) **then O(log(A)+log(I))**
14: borrar((significado(e.itVentasSinPromo, a),i) **O(log(A)+log(I))**
15: **if** #claves(significado(e.itVentasSinPromo, a), i) == 0 **then O(1)**
16: borrar(e.itVentasSinPromo, a) **O(log(A))**
17: **end if**
18: **else**
19: $significado(significado(e.itVentasSinPromo, a), i)$ ← fin(significado(significado(e.itVentasSinPromo, a), i))**O(log(A)+log(I))**
20: **end if**
21: **if** (obtener(e.gastoPorPersona, a) − obtener(e.menu,i)) == 0 **thenO(log(A)+log(I))**
22: borrar(e.gastosPorPersona, a)**O(log(A))**
23: **else**
24: $significado(e.gastoPorPersona, a)$ ← obtener(e.gastoPorPersona, a) − obtener(e.menu,i)**O(log(A)+log(I))**
25: **end if**
26: $significado(p.stock, i)$ ← significado(p.stock, i) + 1 **O(log(I))**

Complejidad: $O(log(A) + log(I))$
Justificacion: Son todas operaciones con complejidad log(A), log(I) ó 1, independientes entre sí.

1.3. Servicios Utilizados por el módulo PuestoDeComida

▷ Módulos que se utilizan, detallando las complejidades, aliasing y otros aspectos que dichos módulos deben proveer para que el módulo actual pueda cumplir con su interfaz.
▷ **Diccionario Logarítmico:**
▷ **Lista Enlazada:**

2. Modulo Lollapatuza

Interfaz

se explica con: LOLLAPATUZA

géneros: `Lollapatuza`.

2.1. Operaciones básicas de Lollapatuza

NUEVOLOLLA(**in** *puestos*: `diccLog(puestoID, puesto)`, **in** *personas*: `Lista(persona)`) → *res* : `Lollapatuza`
Pre ≡ {$\neg\emptyset?(personas) \wedge \neg\emptyset?(claves(puestos) \wedge_L noHayRepetidos(personas)$
$vendenAlMismoPrecio(significados(puestos)) \wedge NoVendieronAun(significados(puestos))$ }
Post ≡ {res $=_{\text{obs}}$ crearLolla(puestos, secuAconj(personas))}
Complejidad: O(1)
Descripción: Genera un Lollapatuza nuevo.
Aliasing: No aplica.

REGISTRARCOMPRA(**in/out** l: `Lollapatuza`, **in** pi: `puestoID`, **in** a: `persona`, **in** i: `item`, **in** c: `cant`)
Pre ≡ { l $=_{\text{obs}}$ $l_0 \wedge a \in personas(l) \wedge def?(pi, puestos(l)) \wedge_L haySuficiente?(obtener(pi, puestos(l)), i, c)$ }
Post ≡ {l = vender(l_0,pi,a,i,c) }
Complejidad: O(Log(A) + log(I) + log(P)+ log(cant))
Descripción: Registra la compra realizada por una persona de cierto item y cantidad en algún puesto.
Aliasing: Se devuelve una referencia modificable al Lollapatuza

HACKEARITEM(**in/out** l: `Lollapatuza`, **in** p: `persona`, **in** i: `item`)
Pre ≡ {l = l_0 ∧ ConsumioSinPromoEnAlgunPuesto(l, a, i)}
Post ≡ {l = hackear(l_0,a,i)}
Complejidad: O(log(I) + log(A) + log(P))
Descripción: Hackear un ítem consumido por una persona. Se hackea el puesto de menor ID en el que la persona haya consumido ese ítem sin promoción.
Aliasing: Se devuelve una referencia modificable al Lollapatuza

GASTOTOTALPERSONA(**in** l: `Lollapatuza`, **in** p: `persona`) → *res* : `nat`
Pre ≡ {p ∈ personas(l)}
Post ≡ {res = gastoTotal(l,p)}
Complejidad: = ($log(A)$)
Descripción: Devuelve el gasto total hecho por una persona en el festival.
Aliasing: Se devuelve una referencia no modificable del valor del gasto total hecho por una persona.

MAYORGASTADOR(**in** l : `Lollapatuza`) → *res* : `persona`
Pre ≡ {$\neg\emptyset?(personas(l))$}
Post ≡ {res = masGasto(l)}
Complejidad: $O(1)$
Descripción: Devuelve la persona que más gastó en el festival, con desempate por menor ID.
Aliasing: Se devuelve una referencia modificable del id del mayor gastador del festival.

PUESTOCONMENORSTOCKDE(**in** l: `Lollapatuza`, **in** i: `item`) → *res* : `puestoID`
Pre ≡ {$\neg\emptyset?(claves(puestos(l))) \wedge_L$ itemEnAlgunMenu(i,significados(puestos(l)))}
Post ≡ {res = menorStock(l,i)}
Complejidad: $O(P * (log(P) + log(I))$
Descripción: Devuelve el puesto con el menor stock de ese item, si es que ese item existe en el menú de algún puesto.
Aliasing: Se devuelve un puestoID por referencia.

OBTENERPERSONAS(**in** l: `Lollapatuza`) → *res* : `Lista(persona)`
Pre ≡ {true}
Post ≡ {secuAConj(res) = personas(l)}
Complejidad: $O(1)$

Descripción: Devuelve el conjunto de personas del festival.
Aliasing: Se devuelve una referencia no modificable a la lista de personas.

ObtenerPuestos(**in** l : `Lollapatuza`) $\rightarrow$ *res* : `dicc(puestoID,puesto)`
Pre $\equiv$ {true}
Post $\equiv$ {res = puestos(l)}
Complejidad: $O(1)$
Descripción: Devuelve el diccionario de puestos con su ID del festival.
Aliasing: Se devuelve una referencia no modificable al diccionario de puestos con su id.

Representación

2.2. Representación del Lollapatuza

`Lollapatuza` **se representa con** `estr`

donde `estr` es `tupla`(*puestos*: `diccLog(puestoID,puesto)`
, *personas*: `Lista(persona)`
, *mayorGastador*: `persona`
, *gastoPorPersona*: `diccLog(persona,dinero)`
, *puestosHackeables*: `diccLog(persona,diccLog(item, diccLog(puestoID, puestoID))`
, *gastos*: `diccLog(dinero, diccLog(persona,persona))`)

1)El festival tiene al menos un puesto

2)No hay personas repetidas

3)Hay al menos una persona en el festival

4)El mayor gastador es la persona de gastoPorPersona cuyo significado es el mayor.En caso de ese ser iguales sus significados, el mayorGastador es quien tiene menor ID.

5)Las claves de gastoPorPersona esta incluidas en personas

6)Si las claves no son vacias en puestosHackeables, entonces las claves del significado tampoco

7)Todos los ID en donde se hizo una compra hackeable estan en puestos

8)Todas las personas que hicieron una compra hackeable estan en gastosPorPersona

9)En gastos, para todo dinero, el diccionario que tiene como significado, tienen sus claves iguales a sus significados

10)En gastos, para todo dinero, las claves de sus significado estan incluidas en claves(gastosPorPersona)

11)claves(gastos) = significados(gastosPorPersona)

12)Para cada clave del significado del significado de puestosHackeables su significado es el mismo.

13)Todas las claves del significado del significado de puestosHackeables son puestos del festival

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$
1)$\neg$Vacio?(claves(e.puestos) $\wedge_L$ ($\forall$p: puestoID)(def?(p, e.puestos) $\Rightarrow_L$ $\neg$Vacio?(obtener(p, e.puestos)))

2)($\forall$ i:nat)(0<i $\leq$ |$e.personas$| $-$ 1 $\rightarrow_L$ e.personas[i] $\neq$ e.personas[i+ 1])

3)$\neg$Vacío?(e.personas)

4)($\exists$ p : persona)(p = e.mayorGastador $\Leftrightarrow$ $\neg$Vacío?(claves(e.gastoPorPersona))$\rightarrow_L$ ($\exists$ p1: persona)(($\forall$ p2:persona)(def?(p1,e.gastosPorPerosna) $\wedge_L$ (p2,e.gastosPorPerosna) $\rightarrow_L$ obtener(p1,e.gastosPorPerosna) >= obtener(p2,e.gastosPorPerosna))) $\wedge$ obtener(p1,gastoPorPersona) = obtener(p2,gastoPorPersona) $\rightarrow$ p1 < p2 $\wedge$ p1 = p)

5)($\forall$ p:persona)(def?(p,e.gastoPorPersona)$\rightarrow_L$ está?(p,e.personas))

6)($\forall$ p:persona)(def?(p,e.puestosHackeables)$\rightarrow_L$ $\neg$ vacío?(claves(obtener(p,e.puestosHackeables))))

7)($\forall$ p:persona)(def?(p,e.puestosHackeables)$\rightarrow_L$
($\forall$ i:item)(def?(i,obtener(p,e.puestosHackeables))$\rightarrow_L$ def?(obtener(i,obtener(p,e.puestosHackeables)),e.puestos))

8)($\forall$ p:persona)(def?(p,e.puestosHackeables)$\rightarrow_L$ def?(p,e.gastoPorPersona))

9)($\forall$ d:dinero)def?(d,e.gastos) $\rightarrow_L$ ($\forall$ p:persona) def?(p, significado(d,e.gastos)) $\rightarrow_L$
significado(p, significado(d,e.gastos)) = p

10)($\forall$d:dinero)def?(d,e.gastos) $\rightarrow_L$ ($\forall$ p:persona) def?(p, significado(d,e.gastos)) $\Leftrightarrow$ def?(p, e.gastoPorPersona)

11)conjuntoAMulticonjunto(e.gasto,claves(e.gasto)) = significados(e.gastosPorPersona)

12)($\forall$ p:persona)($\forall$ i:item)(def?(p,e.puestosHackeables)$\wedge$ def?(i,obtener(p,e.puestosHackeables))$\rightarrow_L$($\forall$ pi:puestoID)
(def?(pi,obtener(i,obtener(p,e.puestosHackeables)))$\rightarrow_L$ pi=$_{\text{obs}}$ obtener(i,obtener(p,e.puestosHackeables))))

13)($\forall$ a:persona)(def?(a,e.puestosHackeables) $\rightarrow_L$ ($\forall$ i:item) def?(i,obtener(a,e.puestosHackeables))$\rightarrow_L$ ($\forall$pi:puestoID)(def?(pi,obtener(i,obtener(a,e.puestosHackeables))$\rightarrow_L$ def?(pi,e.puestos) $\wedge_L$ esHackeable(obtener(e.puestos,pi),a,i)))

2.2.1. Funciones Auxiliares del rep lollapatuza

conjuntoAMulticonjunto(gastos,dineros) =

if vacio?(dineros) **then**

Ø

else

Ag(conjuntoAMulticonjunto(gastos,sinUno(dineros)),
agregarN(dameUno(dineros),#claves(obtener(dameUno(dineros),gastos))))

end if

agregarN(dinero,nat)=

if nat > 0 **then**

Ag(agregarN(dinero,nat-1),Ag(dinero,Ø))

else

Ø

end if

esHackeable(p,a,i)= ¬esVacia?(obtener(obtener(p.itVentasSinPromo), i))

2.2.2. Abs del Lollapatuza

Abs : estr e $\longrightarrow$ lolla {Rep(e)}
Abs(e) $\equiv$ (∀e: estr)Abs(e) $=_{\text{obs}}$ l :lolla
/
e.puestos=puestos(l) ∧ (∀ p:persona)(0≤p < $|e.personas|$ ↔ p∈ personas(l))

Algoritmos

---

iNuevoLolla(**in** *puestos* : `diccLog(puestoID, puesto)`, **in** *personas* : `Lista(persona)`) → *res* : Lollapatuza

1: *res.puestos* ← puestos **O(1)**
2: *res.personas* ← personas **O(1)**
3: *res.mayorGastador* ← 0 **O(1)**
4: *res.gastoPorPersona* ← dicc::Vacío() **O(1)**
5: *res.puestosHackeables* ← dicc::Vacío() **O(1)**
6: *res.gastos* ← dicc::Vacío() **O(1)**
7: *res.itGastos* ← dicc::Vacío() **O(1)**

Complejidad: $O(1)$
Justificación: Como son todas asignaciones por referencia no se realiza el copiado ▷

---

iRegistrarCompra(**in/out** l : `Lollapatuza`,**in** pi : `puestoID`,**in** a : `persona`,**in** i : `item`,**in** c : `cant`)

1: $p \leftarrow$ significado(l.puestos, pi) **O(log(P))**
2: Venta(p, a, i, c) **O(log(I)+ log(A) + log(cant)**

3: **if** definido(l.gastosPorPersona, a) **then O(log(A))**
4: $gastoAnt \leftarrow$ significado(l.gastoPorPersona,a) **O(log(A))**
5: $gastoNuevo \leftarrow$ signficado(l.gastoPorPersona,a)+((significado(p.menu, i) × c) − ((significado(p.menu, i)× c) × ObtenerDescuento(p,i,c))) **O(log(A)+log(I)+log(cant))**

6: $signficado(l.gastoPorPersona, a) \leftarrow$ gastoNuevo **O(log(A))**

7: borrar(significado(l.gastos, gastoAnt),a) **O(log(A))**
8: **if** definido?(l.gastos, gastoNuevo) **then O(log(A))**
9: definir(significado(l.gastos, gastoNuevo),a,a) **O(log(A)**
10: **else**
11: definir(l.gastos, gastoNuevo, vacio()) **O(log(A))**
12: definir(significado(l.gastos, gastoNuevo),a,a) **O(log(A))**
13: **end if**
14: **else**
15: $gastoNuevo \leftarrow$ (significado(p.menu, i)× c) − ((significado(p.menu, i)× c) × (ObtenerDescuento(p,i,c) / 100)) **O(log(I)+log(cant))**

16: definir(l.gastoPorPersona, a, gastoNuevo) **O(log(A))**

17: **if** definido?(l.gastos, gastoNuevo) **then O(log(A))**
18: definir(significado(l.gastos, gastoNuevo),a,a) **O(log(A))**
19: **else**
20: definir(l.gastos, gastoNuevo, vacio()) **O(log(A))**
21: definir(significado(l.gastos, gastoNuevo),a,a) **O(log(A))**
22: **end if**
23: **end if**

24: **if** a != l.mayorGastador & gastoNuevo > significado(l.GastoPorPersona,l.mayorGastador) **then O(log(A))**
25: $l.mayorGastador \leftarrow$ a **O(1)**
26: **end if**

27: **if** ObtenerDescuento(p,i,c) == 0 **then O(log(I)+log(cant))**
28: **if** definido?(l.puestoHackeable,a) **then O(log(A))**
29: **if** definido?(significado(l.puestoHackeable,a),i) **then O(log(A)+log(I))**
30: **if** ¬definido?(significado(significado(l.puestoHackeable,a),i),pi) **then O(log(A)+log(I)+log(P))**
31: definir(significado(significado(l.puestoHackeable,a),i),pi,pi) **O(log(A)+log(I)+log(P))**
32: **end if**
33: **else**
34: definir(significado(l.puestoHackeable,a),i, vacio()) **O(log(A)+log(I))**
35: definir(significado(significado(l.puestoHackeable,a),i),pi,pi) **O(log(A)+log(I)+log(P))**
36: **end if**
37: **else**
38: definir(l.puestoHackeable, a, vacio()) **O(log(A))**
39: definir(significado(l.puestoHackeable,a),i, vacio()) **O(log(A)+log(I))**
40: definir(significado(significado(l.puestoHackeable,a),i),pi,pi) **O(log(A)+log(I)+log(P))**
41: **end if**
42: **end if**

Complejidad: $O(log(A) + log(I) + log(P) + log(cant))$
Justificación: Realizar operaciones en el diccionario gastos tiene complejidad $O(log(dineros))$ siendo dineros la cantidad de gastos definidos, pero como hay maximo un gasto distinto para cada persona, cantidad de gastos es menor igual a cantidad de personas, por lo tanto log(dineros) ∈ O(log(A)). El resto de operaciones toman log(A), log(I) ó log(P).

iHackear(**in/out** l : `lollapatuza`, **in** a : `persona`, **in** i : `item`)

1: $it \leftarrow$ CrearIt(significado(significado(l.puestosHackeables,a),i)) **O(1)**
2: $pi \leftarrow$ siguienteClave(it) **O(1)**
3: $p \leftarrow$ significado(l.puestos, pi) **O(log(P))**
4: hackearP(p, a, i) **O(log(A)+log(I))**
5: **if** definido(p.itVentasSinPromo,a) **then O(log(A))**
6: **if** ¬definido(significado(p.itVentasSinPromo,a),i) **then O(log(A)+log(I))**
7:
8: borrar(significado(significado(l.puestosHackeables,a),i), pi) **O(log(A)+log(I)+log(P))**
9: **end if**
10: **else**
11: borrar(significado(significado(l.puestosHackeables,a),i), pi) **O(log(A)+log(I)+log(P))**
12: **end if**
13: **if** #claves(significado(significado(l.puestosHackeables,a),i)) == 0 **then**
14: borrar(significado(l.puestosHackeables,a),i)
15: **if** #claves(significado(l.puestosHackeables,a)) == 0 **then**
16: borrar(l.puestosHackeables,a)
17: **end if**
18: **end if**
19: $gastoAnt \leftarrow$ significado(l.gastoPorPersona,a)
20: $gastoNuevo \leftarrow$ signficado(l.gastoPorPersona,a) − significado(p.menu,i) **O(log(A)+log(I))**
21: borrar(significado(l.gastos, gastoAnt),a) **O(log(A)**
22: **if** gastoNuevo ! = 0 **then O(1)**
23: $signficado(l.gastoPorPersona, a) \leftarrow$ gastoNuevo **O(log(A))**
24: **if** definido?(l.gastos, gastoNuevo) **then O(log(A))**
25: definir(significado(l.gastos, gastoNuevo),a,a) **O(log(A))**
26: **else**
27: definir(l.gastos, gastoNuevo, Vacio()) **O(log(A))**
28: definir(significado(l.gastos, gastoNuevo),a,a) **O(log(A))**
29: **end if**
30: **else**
31: borrar(l.gastoPorPersona, a)
32: **end if**
33: **if** a == l.mayorGastador **then O(1)**
34: **if** #claves(significado(l.gastos, gastoAnt)) != 0 **then O(1)**
35: $it \leftarrow$ CrearIt(significado(l.gastos, gastoAnt)) **O(log(A))**
36: $l.mayorGastador \leftarrow$ siguienteClave(it) **O(1)**
37: **else**
38: borrar(l.gastos, gastoAnt) **O(log(A))**
39: $it \leftarrow$ CrearItUlt(l.gastos) **O(1)**
40: $2gasto \leftarrow$ AnteriorClave(it) **O(1)**
41: $it2 \leftarrow$ CrearIt(significado(l.gastos, 2gasto)) **O(1)**
42: $l.mayorGastador \leftarrow$ SiguienteClave(it2) **O(1)**
43: **end if**
44: **end if**

Complejidad: $O(log(A) + log(P) + log(I))$
Justificación: Asumimos que CrearItUlt crea un iterador al ultimo elemento del diccionario, es decir, al de mayor clave y CrearIt crea un iterador al primer elemento del diccionario, es decir, al de menor clave. En el caso de que el puesto deje de ser hackeable se suma la complejidad log(P) dado que hay que eliminarlo del diccionario PuestosHackeables. El resto son todas operaciones con complejidad log(I) ó log(A).

iGastoTotalPersona(**in** l : `lollapatuza`, **in** p : `persona`) → res : nat

```
1: res ← significado(l.gastoPorPersona,p)
```

Complejidad: $O(log(a))$

Justificación: la operacion significado en un diccLog cuesta log(#claves)

---

iMayorGastador(**in** l : `lollapatuza`) → res : persona

```
1: res ← l.mayorGastador
```

Complejidad: $O(1)$

Justificación: cuesta $O(1)$ porque la información esta en la estructura

---

iPuestoConMenorStock(**in** l : `lollapatuza`,**in** i : `item`) → res : puestoID

```
1: res← 0
2: for (pi definido?(l.puestos, pi))) do O(P)
3:     if res== 0 then
4:         if definido(significado(l.puestos, pi).stock,i) then O(log(p) + log(i))
5:             res ← pi 0(1)
6:         end if
7:     else
8:         if definido(significado(l.puestos, pi).stock,i)  significado(significado(l.puestos, pi).stock,i) < significado(pi.sotck,i) then
9:             res ← pi
10:        else
11:            if definido(significado(l.puestos, pi).stock,i)  significado(significado(l.puestos, pi).stock,i) == significado(pi.sotck,i)  pi < res then O(log(p) + log(i))
12:                res ← pi
13:            end if
14:        end if
15:
16:        if res== 0 then
17:            it ← CrearIt(l.puestos) O(1)
18:            res ← siguienteClave(it) O(1)
19:        end if
```

Complejidad: $O(P(log(P) + log(I)))$

Justificación: Asumimos que CrearIt crea un iterador al primer elemento del diccionario, es decir, al de menor clave. En cuanto a la complejidad que queda, el for recorre todos los puestos (esto tarda O(P)) y dentro del for se utilizan operaciones del diccLog las cuales cuestan log(# claves).

---

iObtenerPersonas(**in** l : `lollapatuza`) → res : Lista(persona)

```
1: res← l.personas
```

Complejidad: O(1)

Justificación: cuesta O(1) porque la información esta en la estructura =0

---

iObtenerPuestos(**in** l : `lollapatuza`) → res : dicc(puestoID,puesto)

```
1: res← l.puestos
```

Complejidad: O(1)

Justificación: cuesta 0(1) porque la información esta en la estructura

---

2.3. Servicios Utilizados por el módulo Lollapatuza

▷ Módulos que se utilizan, detallando las complejidades, aliasing y otros aspectos que dichos módulos deben proveer para que el módulo actual pueda cumplir con su interfaz.
▷ **Diccionario Logarítmico:**
▷ **Lista Enlazada:**
▷ **Puesto De Comida:**