Algoritmos y Estructura de Datos 2

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

Trabajo práctico 2: Modularizacion

Lollapatuza

CCC

Integrante	LU	Correo electrónico
Francisco Ignacio Cueto	223/22	francue3@gmail.com
Santiago Rivas Molinari	415/22	santiagorivas0203@gmail.com
Felipe Santiago Martin	542/22	felipe.martin2002@gmail.com
Francisco Lingua	68/22	franciscolingua945@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega	*Maximiliano Martino*	Aprobado (-)
Segunda entrega		

Correcciones generales (Aprobado condicional):
Si bien el TP esta aprobado, para el TP3 va a ser condicion para aprobar:
- Escoger nombres representativos y coherentes para lo que estan haciendo
- Agregar comentarios en todas las operaciones cuando no son triviales
- Modularizar algoritmos complejos en funciones auxiliares con nombres representativos

1 Interfaz Lollapatuza

Interfaz

NUEVOLOLLA(in puestos : dicc(IDPuesto, puesto), in personas: conj(DNI)) $\rightarrow$ *res* : `lolla`

Pre $\equiv$ {true}

Post $\equiv$ {res= crearLolla(ps, as)}

Complejidad: $O(A \times Log(A) + P \times (I \times (Log(I) + Log(P))))$

Descripción: crea una instancia del Lollapatuza

Aliasing: puestos, personas y lolla son pasados por referencia

VENTA(in/out l: lolla, in i: item, in a: DNI, in p: IDPuesto, in c: nat)

Pre $\equiv$ {a $\in$ personas(l) $\wedge$ def?(pi, puestos(l)) $\wedge$ i $\in$ menu(p) $\wedge$ haySuficiente?(obtener(p, puestos(l)), i, c) $\wedge$ $l_0 = l$}

Post $\equiv$ {l = vender(l_0, p, a, i, c)}

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

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

Aliasing: lolla es pasado como una referncia modificable, i, a, c y p son pasados por copia

HACKEAR(in/out l: lolla, in i: item, in a: DNI)

Pre $\equiv$ {ConsumioSinPromoEnAlgunPuesto(l, a, i) $\wedge$ l_0 = l}

Post $\equiv$ {hackearMinimo(i,a,l_0) = l}

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

Descripción: hackea un item consumido por una persona en el puesto valido con menor ID

Aliasing: lolla se pasa como referencia modificable, el item y el dni son pasados por copia

GASTODEPERSONA(in l: lolla, in p: DNI) $\rightarrow$ *res* : `dinero`

Pre $\equiv$ {p $\in$ personas(l)}

Post $\equiv$ {res = gastoTotal(p)}

Complejidad: $O(log(A))$

Descripción: Se obtiene el gasto total de una persona

Aliasing: lolla se pasa como referencia no modificable y tanto el dni como res son pasados por copia

MAYORGASTADOR(in L: lolla) → *res* : `DNI`
Pre ≡ {¬(Vacío?(personas(L)))}
Post ≡ {res = masGasto(L)}
Complejidad: $O(1)$
Descripción: Se obtiene el DNI de la persona que más gasta y menor ID
Aliasing: Se pasa L por referencia no modificable y el res por copia

PUESTOCONMENORSTOCK(in L: Lolla,in I: Item) → *res* : `IDPuesto`
Pre ≡ {P ∈ puestos(L)}
Post≡ {res = menorStock(L, I)}
Complejidad: $O(log(I) + P)$
Descripción: se obtiene el puesto con menor stock
Aliasing: Se pasa L por referencia no modificable, I es pasado por copia y res es devuelto por copia

PERSONAS(in l: lolla) → *res* : `conj(DNI)`
Pre ≡ {true}
Post ≡ {res = personas(l)}
Complejidad: $O(1)$
Descripción: devuelve las personas del lolla
Aliasing: lolla se pasa por referencia no modificable y res se pasa por referencia no modificable

PUESTOS(in l: lolla) → *res* : `dic(IDPuesto, puesto)`
Pre ≡ {true}
Post ≡ {res = puestos(l)}
Complejidad: $O(1)$
Descripción: devuelve los puestos del lolla
Aliasing: lolla se pasa por referencia no modificable y res se pasa por referencia no modificable

2 Interfaz Puesto

Interfaz

2.1 Operaciones Basicas

NUEVOPUESTO(in p: dicc(item, nat), in s: dicc(item, nat), in d: dicc(item, dicc(cant, nat))) $\rightarrow$ *res* : `puesto`
Pre $\equiv$ {claves(p) = claves(s) $\wedge$ claves(d) $\subseteq$ claves(p)}
Post $\equiv$ {res = crearPuesto(p,s,d)}
Complejidad: $O(I \times log(I))$
Descripción: Crea un nuevo puesto de comida
Aliasing: recibe todos los diccionarios por referencia no modificable y res se pasa por copia

STOCK(In p : Puesto, In i : Item) $\rightarrow$ *res* : `Nat`
Pre $\equiv$ {i $\in$ Menu(p)}
Post $\equiv$ {res = stock(p,i)}
Complejidad: $O(log(p))$
Descripción: Se devuelve el stock del item en el puesto pasado por parametro
Aliasing: recibe un puesto por referencia no modificable y un item por copia y devuelve un parametro por copia

DESCUENTODEITEM(in l: puesto, in i: item, in cant: nat) $\rightarrow$ *res* : `nat`
Pre $\equiv$ {i $\in$ Menu(l) $\wedge$ cant $>$ 0}
Post $\equiv$ {res = descuento(l,i,cant)}
Complejidad: $O(log(I) + log(cant))$
Descripción: se devuelve el descuento del item segun la cantidad comprada
Aliasing: l es pasado por referencia no modificable, i y cant son pasados por copia y me devuelve un res que es una referencia no modificable

CUANTOGASTO(in p : puesto, in per: DNI) $\rightarrow$ *res* : `dinero`
Pre $\equiv$ {true}
Post $\equiv$ {res = GastosDe(p,per)}
Complejidad: $O(log(A))$
Descripción: se devuelve lo que gasto una persona en un puesto determinado
Aliasing: el puesto se pasa por referencia no modificable y tanto el dni como res estan por copia

3 Representación

Representación

3.1 Estructura Lolla

`Lollapatuza` **se representa con** `lolla`

donde `lolla` es `tupla`(*Personas*: `diccLog(DNI, persona)`,
ConjDNIs: `conj(DNI)`,
Puestos: `diccLog(IDPuesto, puesto)`,
MaximoGastador: `Max`,
Gastos: `diccLog(nat, conjLog(DNI))` ,
Items: `diccLog(item, diccLog(IDPuesto, itDicc(IDPuesto, puesto))))`)

donde `persona` es `tupla`(*GastoTotal*: `nat`,
Compras: `(DiccLog(item, NodoCompras))`)

donde `NodoCompras` es `tupla`(*PuestoHack*: `itDicc(IDPuesto,Hack)` ,
Dicc: `diccLog(IDPuesto,Hack)`)

donde `Hack` es `tupla`(*puesto*: `itDicc(IDPuesto, puesto)` ,
cantHackeable: `nat`)

donde `Max` es `tupla`(*it*: `itDicc(nat, conjLog(DNI))` , *id*: `DNI`)

3.2 Estructura Puesto

`PuestoDeComida` **se representa con** `puesto`

donde `puesto` es `tupla`(*items*: `diccLog(item, infoItem)`,
clientes: `diccLog(DNI, gasto : nat)`)

donde `infoItem` es `tupla`(*stock*: `nat`,
precio: `nat`,
dtos: `diccLog(nat, nat)`)

3.3 Invariante de Representacion Lolla

Rep : Lolla $\longrightarrow$ bool

✓ Rep(l) $\equiv$ true $\iff$ $MaxGastador(l) \land MaxGastadorIT(l) \land PersonasConUnMismoGasto(l) \land MismosDNI(l) \land DNIConUnSoloGasto(l) \land CantidadDeGastosValida(l) \land ClienteValido(l) \land GastoEstaEnLolla(l) \land ItemCompradoEnLolla(l) \land ItemEsVendidoPorUnPuestoExistente(l) \land itItems(l) \land VendenAlMismoPrecio(l) \land GastoTotalDePersona(l) \land MismosIDs(l) \land IDHackValida(l) \land CantHackValida(l)$

3.4 Funcion de Abstraccion Lolla

Abs : lolla e $\longrightarrow$ Lolla {Rep(e)}

✓ Abs(e) $=_{obs}$ l: Lolla | puestos(l) = e.puestos $\land$
personas(l) = e.ConjDNIs

3.5 Invariante de Representacion Lolla - Auxiliares

MaxGastador : Lolla $\longrightarrow$ Bool

✓ MaxGastador(l) $\equiv$ ($\exists$ Max: DNI)($\forall$ dni : DNI)(Max $\in$ claves(l.Persona) $\land$
Max = l.MaximoGastador.id $\land$ dni $\in$ claves(l.Persona) $\land$
dni != Max $\Rightarrow_{L}$
(obtener(dni,l.personas).GastoTotal<
obtener(esMax,l.personas).GastoTotal $\lor$
(dni > Max $\land$
obtener(dni,l.personas).GastoTotal=
obtener(esMax,l.personas).GastoTotal)))

✓ MaxGastadorIT : Lolla $\longrightarrow$ Bool
MaxGastadorIT(l) $\equiv$ siguienteClave(l.MaxGastador.it) = l.MaxGastador.id$\land$
siguienteSignificado(l.MaxGastador.it)=
obtener(l.MaxGastador.id,l.puestos)

✓ PersonasConUnMismoGasto : Lolla $\longrightarrow$ Boolean
PersonasConUnMismoGasto(l) $\equiv$ ($\forall$ g: nat)(g $\in$ claves(l.gastos) $\Rightarrow_{L}$
obtener(g,l.gastos) $\subseteq$ claves(l.Personas)

✓ MismosDNI : Lolla $\longrightarrow$ Boolean
MismosDNI(l) $\equiv$ l.ConjDNIs = claves(l.Personas)

DNIConUnSoloGasto : Lolla $\longrightarrow$ Boolean
DNIConUnSoloGasto(l) $\equiv$ ($\forall$ g: nat)($\forall$ dni: DNI)
(dni $\in$ obtener(g,l.gastos) $\Rightarrow_{\text{L}}$
$\neg$($\exists$ g': nat)
(g' ! = g $\wedge_{\text{L}}$ dni $\in$ obtener(g',l.gastos)))

CantidadDeGastosValida : Lolla $\longrightarrow$ Boolean
CantidadDeGastosValida(l) $\equiv$ claves(l.Gastos) $\leq$ claves(l.Personas)

ClienteValido : Lolla $\longrightarrow$ Boolean
ClienteValido(l) $\equiv$ ($\forall$ p:puesto)($\forall$ id:IDPuesto)(definido?(id,l.puestos) $\Rightarrow_{\text{L}}$
p = obtener(id,l.puestos) $\Rightarrow_{\text{L}}$ ($\forall$ dni: DNI)
(dni $\in$ claves(p.clientes) $\Rightarrow_{\text{L}}$ dni $\in$ claves(l.Personas)))

GastoEstaEnLolla : Lolla $\longrightarrow$ Boolean
GastoEstaEnLolla(l) $\equiv$ ($\forall$ p:DNI)(definido?(p,l.personas) $\Rightarrow_{\text{L}}$
obtener(p,l.personas).GastoTotal$\in$claves(l.gastos)$\Rightarrow_{\text{L}}$
p $\in$ obtener(obtener(p,l.personas).GastoTotal,l.gastos))

ItemCompradoEnLolla : Lolla $\longrightarrow$ Boolean
ItemCompradoEnLolla(l) $\equiv$ ($\forall$ p: DNI)($\forall$ i:item)
(i $\in$ clave(significado(significado(p,l.personas).Compras))$\Rightarrow_{\text{L}}$
i $\in$ clave(l.items))

ItemEsVendidoPorUnPuestoExistente : Lolla $\longrightarrow$ Boolean
ItemEsVendidoPorUnPuestoExistente(l) $\equiv$ ($\forall$ i: item)($\forall$ IdP:nat)
((definido?(i,l.items) $\Rightarrow_{\text{L}}$
IdP $\in$ clave(obtener(i,l.items)))
$\longleftrightarrow$ IdP $\in$ clave(l.Puestos)$\Rightarrow_{\text{L}}$ i $\in$
clave(obtener(IdP,l.puestos).items))

itItems : Lolla $\longrightarrow$ Boolean
itItems(l) $\equiv$ ($\forall$ i : item)(definido?(i,l.items) $\Rightarrow_{\text{L}}$
($\forall$ IdP: IDPuesto) (definido?(IdP, obtener(i,l.items)) $\Rightarrow_{\text{L}}$
($\forall$ it : itDicc(IDPuesto, puesto))
(it = obtener(IdP, obtener(i,l.items)) $\Rightarrow_{\text{L}}$
IdP = siguienteClave(it) $\wedge$ siguienteSignificado(it) =
obtener(IdP,l.puestos))))

VendenAlMismoPrecio : Lolla $\longrightarrow$ Boolean
VendenAlMismoPrecio(l) $\equiv$ ($\forall$ p,p':puesto)($\forall$ id, id' : IDPuesto)
((id!=id' $\wedge$ p!=p' $\wedge$ definido?(id,l.puestos) $\wedge$
definido(id',l.puestos) $\Rightarrow_{\text{L}}$
p = obtener(id, l.puestos) $\wedge$
p' = obtener(id', l.puestos)) $\Rightarrow_{\text{L}}$
($\forall$ i : item)
((obtener(i,p.items)).precio=(obtener(i,p'.items)).precio))

GastoTotalDePersona : Lolla $\longrightarrow$ Boolean
GastoTotalDePersona(l) $\equiv$ ($\forall$ a : DNI)($\forall$ id:IDPuestos)($\forall$ p : puesto)
(a $\in$ claves(l.personas) $\wedge$ id $\in$ claves(l.puestos)$\Rightarrow_{\text{L}}$
p = obtener(id,l.puestos)$\Rightarrow_{\text{L}}$
significado(l.personas,a).GastoTotal =
$\sum$ (if def?(p.clientes, a) then obtener(a,p.clientes)
else 0 fi))

MismosIDs : Lolla $\longrightarrow$ Boolean
MismosIDs(l) $\equiv$ ($\forall$ p: DNI)(p $\in$ claves(l.personas) $\wedge_{\text{L}}$
($\forall$ i: item)(i $\in$ claves(obtener(i,l.items)) $\Rightarrow_{\text{L}}$
claves(obtener(i,obtener(p,l.personas).compras).dicc)$\subseteq$claves(l.puestos)))

DefComprasHack? : Lolla $\times$ DNI $\times$ Item $\times$ IDPuesto $\longrightarrow$ Boolean
DefComprasHack?(l,p,i,ID) $\equiv$ definido?(ID,(obtener(i,obtener(p,l.personas).compras).Dicc))

SigComprasHack : Lolla $\times$ DNI $\times$ Item $\times$ IDPuesto $\longrightarrow$ <itDicc(IDPuesto,puesto), nat>
SigComprasHack(l,p,i,ID) $\equiv$ obtener(ID, (obtener(i,obtener(p,l.personas).compras).Dicc))

CantHackValida : Lolla $\longrightarrow$ Boolean
CantHackValida(l) $\equiv$ ($\forall$ p: DNI)(definido?(p, l.personas) $\wedge_{\text{L}}$
($\forall$ i: item)((definido?(i,obtener(p,l.personas).compras))$\wedge_{\text{L}}$
($\forall$ ID : IDPuesto)(DefComprasHack?(l,p,i,ID) $\Rightarrow_{\text{L}}$
definido?(p,siguienteSignificado(SigComprasHack(l,p,i,ID).puesto).clientes)
$\wedge_{\text{L}}$
obtener(p,siguienteSignificado(SigComprasHack(l,p,i,ID).puesto).clientes)
$\geq$ $\sum$ SigComprasHack(l,p,i,ID).cantHackeable $\times$
obtener(i,siguienteSignificado(SigComprasHack(l,p,i,ID).puesto).items).precio)

IDHackValida : Lolla $\longrightarrow$ Boolean
IDHackValida(l) $\equiv$ ($\forall$ p: DNI)(definido?(p, l.personas) $\wedge_{\text{L}}$
($\forall$ i: item)((definido?(i, obtener(p,l.personas).compras))
$\Rightarrow_{\text{L}}$ ($\exists$ ID : IDPuesto)(DefComprasHack?(l,p,i,ID)) $\Rightarrow_{\text{L}}$
($\forall$ ID' : IDPuesto)(DefComprasHack?(l,p,i,ID') $\wedge$
ID != ID' $\Rightarrow_{\text{L}}$ ID < ID'))$\Rightarrow_{\text{L}}$
($\forall$ it : itDicc(IDPuesto,Hack))
(it = obtener(i, obtener(p,l.personas).compras).PuestoHack$\Rightarrow_{\text{L}}$
siguienteClave(it) = ID $\wedge$
siguienteSignificado(it) = SigComprasHack(l,p,i,ID))))

✓ IDHackEnPuntero : Lolla $\longrightarrow$ Boolean
IDHackEnPuntero $\equiv$ ($\forall$ p: DNI, i: item)(definido?(l.Personas, p) $\wedge_L$
definido?(obtener(l.Personas,p).Compras, i) $\Rightarrow_L$
($\forall$ ID : nat)(DefComprasHack?(l,p,i,ID) $\Rightarrow_L$
definido?(l.Puestos, ID) $\wedge_L$
SiguienteClave(SigComprasHack(l,p,i,ID).puesto)=ID$\wedge_L$
SiguienteSignificado(SigComprasHack(l,p,i,ID).puesto)=obtener(l.Puestos,ID)))

3.6 Invariante de Representacion Puesto

Rep : Puesto $\longrightarrow$ bool
✓ Rep(p) $\equiv$ true $\Longleftrightarrow$ DtosValidos(p) $\wedge_L$
($\forall$ i: item) (i $\in$ claves(p.items) $\Rightarrow_L$ ($\forall$ c: nat) c $\in$ claves(obtener(i, p.items).dtos) $\longleftrightarrow$ c $>$ 0) $\wedge_L$
($\forall$ d: DNI) (d $\in$ claves(p.clientes) $\Rightarrow_L$ obtener(d, p.clientes) $>$ 0) $\wedge_L$
($\forall$ i: item) (i $\in$ claves(p.items) $\Rightarrow_L$ obtener(i, p.items).precio ¿ 0)

3.7 Funcion de Abstraccion Puesto

Abs : puesto e $\longrightarrow$ PuestoDeComida {Rep(e)}
✓ Abs(e) $=_{obs}$ p: PuestoDeComida | menu(p) = claves(e.items) $\wedge_L$
($\forall$ i: item)(i $\in$ Claves(e.items) $\Rightarrow_L$
obtener(i, e.items).stock = stock(p,i) $\wedge$
obtener(i, e.items).precio = precio(p,i) $\wedge$
($\forall$ c: nat)(c $\in$ Claves(obtener(i, e.items).dtos)
$\Rightarrow_L$
obtener(c, obtener(i, e.items).dtos)= descuentos(p,i,c))) $\wedge$
($\forall$ a: DNI)(a $\in$ Claves(e.clientes) $\Rightarrow_L$ obtener(a, e.clientes) = gastoDe(p,a))

3.8 Invariante de Representacion Puesto - Auxiliares

DtosValidos : PuestoDeComida $\longrightarrow$ Bool
✓ DtosValidos(p) $\equiv$ ($\forall$ i : item)(i $\in$ claves(p.items) $\Rightarrow_L$ ($\forall$ c : cant) c $\in$ claves(obtener(i, p.items).dtos) $\Rightarrow_L$ obtener(obtener(i, p.items).dtos, c) ¡ 100

✗ - Agregar comentarios descriptivos
- Elegir mejor nombre de variables

4 Algoritmos

4.1 Axiomas Lolla

✓iNuevoLolla(**in** *puestos*: `dicc(puestoID,puesto)`,**in** *personas*: `conj(DNI)`) $\rightarrow$ *res* : lolla

1: $itPersona \leftarrow CrearIt(personas)$ ▷ $\Theta(1)$
2: $diccPersona \leftarrow Vacio()$ ▷ $\Theta(1)$
3: $ConjDNIs \leftarrow personas$ ▷ $O(A)$
4: **while** $HaySiguiente(itPersona)$ **do** ▷ $O(A)$
5: $Definir(diccPersona, Siguiente(itPersona), < 0, Vacio() >$ ▷ $O(Log(A))$
6: $Avanzar(itPersona)$ ▷ $\Theta(1)$
7: **end while**
8: $diccItem \leftarrow Vacio()$ ▷ $\Theta(1)$
9: $itPuesto \leftarrow CrearIt(puestos)$ ▷ $\Theta(1)$
10: **while** $HaySiguiente(itPuesto)$ **do** ▷ $O(P)$
11: $itItemPuesto \leftarrow CrearIt(SiguienteSignificado(itPuestos).items)$ ▷ $\Theta(1)$
12: **while** $HaySiguiente(itItemPuesto)$ **do** ▷ $O(I)$
13: $tuplaItem \leftarrow < SiguienteClave(itPuesto), SiguienteSignificado(itItemPuesto) >$ ▷ $\Theta(1)$
14: **if** $Definido?(diccItem, SiguienteClave(itItemPuesto))$ **then** ▷ $O(Log(I))$
15: $diccionarioInterno \leftarrow Significado(diccItem, SiguienteClave(itItemPuesto))$ ▷ $O(Log(I))$
16: $Definir(diccionarioInterno, SiguienteClave(itPuesto), itPuesto)$ ▷ $O(Log(P))$
17: **else**
18: $Definir(diccItem, SiguienteClave(itItemPuesto), Vacio())$ ▷ $O(Log(I))$
19: $diccionarioInterno \leftarrow Significado(diccItem, SiguienteClave(itItemPuesto))$ ▷ $O(Log(I))$
20: $Definir(diccionarioInterno, SiguienteClave(itPuesto), itPuesto)$ ▷ $O(Log(P))$
21: **end if**
22: $Avanzar(itItemPuesto)$ ▷ $\Theta(1)$
23: **end while**
24: $Avanzar(itPuesto)$ ▷ $\Theta(1)$
25: **end while**
26: $diccGasto \leftarrow Vacio()$ ▷ $\Theta(1)$
27: $max \leftarrow Definir(diccGasto, 0, personas)$ ▷ $O(log(A))$
28: $itMinimoDNI \leftarrow CrearIt(diccPersonas)$ ▷ $\Theta(1)$
29: $maximo \leftarrow$<max, SiguienteClave(itMinimoDNI)> ▷ $\Theta(1)$
30: $res \leftarrow$<puestos, diccPersona, diccItem, diccGasto, maximo> ▷ $\Theta(1)$

✓ Complejidad: $O(A \times Log(A) + P \times (I \times (Log(I) + Log(P)))$
Justificación: Define a todas las personas del conjunto en el nuevo diccionario, recorrer todo el conjunto es $O(A)$ y definir en el diccionario es $O(Log(A))$. Luego recorre todos los items de un puesto para definirlo en el diccionario de items y dentro del mismo definir un nuevo diccionario de puestos; hace esto para todos los puestos. Recorrer todos los items es $O(I)$, definir un elemento en el diccionario logaritmico es $O(Log(I))$ y hacerlo para el diccionario interno es $O(Log(P))$ recorrer todos los puestos es $O(P)$; se deben multiplicar estas 4 complejidades para resultar en $O(P \times (I \times (Log(I) + Log(P))))$

✗ - Agregar comentarios descriptivos
- Elegir mejor nombre de variables

✓ iVenta(**in/out** l : `Lolla`, **in** i : `Item`, **in** a : `DNI`, **in** p : `IDPuesto` **in** c : `nat`)

1: $PuestoVenta \leftarrow Significado(p, l.Puestos)$ ▷ $O(Log(P))$
2: $ItemEnPuesto \leftarrow Significado(i, PuestoVenta.items)$ ▷ $O(Log(I))$
3: $ItemEnPuesto.stock \leftarrow ItemEnPuesto.stock - c$ ▷ $\Theta(1)$
4: $Dto \leftarrow DescuentoDeItem(PuestoVenta, i, c)$ ▷ $O(log(I) + log(cant))$
5: $GastoCompra \leftarrow ItemEnPuesto.precio \times \frac{100-Dto}{100} \times c$ ▷ $\Theta(1)$
6: **if** $Definido?(PuestoVenta.clientes, a)$ **then** ▷ $O(log(A)$
7: $Definir(PuestoVenta.clientes, a, Significado(PuestoVenta.clientes, a)+ GastoCompra)$ ▷ $O(log(A))$
8: **else**
9: $Definir(PuestoVenta.clientes, a, GastoCompra)$ ▷ $O(log(A))$
10: **end if**
11: $Persona = Significado(l.personas, a)$ ▷ $O(log(A))$
12: $GastoTotalAnt = Persona.GastoTotal$ ▷ $\Theta(1)$
13: $Persona.GastoTotal = Persona.GastoTotal + GastoCompra$ ▷ $\Theta(1)$
14: $NuevoTotal = Persona.GastoTotal$ ▷ $\Theta(1)$
15: $ItMax = l.MaxGastador.it$ ▷ $\Theta(1)$
16: **if** $SiguienteClave(ItMax) = NuevoTotal$ **then** ▷ $\Theta(1)$
17: $Agregar(SiguienteSignificado(ItMax), a)$ ▷ $\Theta(1)$
18: **if** $a < l.MaximoGastador.ID$ **then** ▷ $\Theta(1)$
19: $l.MaximoGastador.ID = a$
20: **end if**
21: **else**
22: **if** $SiguienteSignificado(ItMax) < NuevoTotal$ **then**
23: $it \leftarrow definir(l.gastos, NuevoTotal, Agregar(Vacio(), a))$ ▷ $O(log(A))$
24: $l.MaxGastador.it = it$ ▷ $\Theta(1)$
25: $l.MaxGastador.ID = a$ ▷ $\Theta(1)$
26: **else**
27: **if** $Definido?(l.gastos, NuevoTotal)$ **then**
28: $Agregar(Significado(l.gastos, NuevoTotal), a)$ ▷ $O(log(A))$
29: **else**
30: $Definir(l.gastos, NuevoTotal, Agregar(Vacio(), a))$ ▷ $O(log(A))$
31: **end if**
32: **end if**
33: **end if**
34: **if** $size(Significado(l.gastos, GastoTotalAnt)) = 1$ **then** ▷ $O(log(A))$
35: $Borra(l.gastos, GastoTotalAnt)$ ▷ $O(log(A))$
36: **else**
37: $Borra(Significado(l.gastos, GastoTotalAnt), a)$ ▷ $O(log(A))$
38: **end if**

39: **if** $Dto = 0$ **then**
40: **if** $Definido?(persona.Compras, i)$ **then** ▷ $O(log(I))$
41: $itLocal \leftarrow Significado(Significado(l.Items, i), p)$ ▷ $O(log(I) + O(log(P))$
42: $ItemNuevoDicc = Vacio()$ ▷ $\Theta(1)$
43: $itDicc \leftarrow Definir(ItemNuevoDicc, p, < itLocal, c >)$ ▷ $\Theta(1)$
44: $ItemNuevo \leftarrow < itDicc, ItemNuevoDicc >$ ▷ $\Theta(1)$
45: $Definir(persona.Compras, i, ItemNuevo)$ ▷ $O(log(I))$
46: **else**
47: **if** $Definido?(Significado(persona.Compras, i).dicc, p)$ **then** ▷ $O(log(I) + O(log(P))$
48: $Significado(Significado(persona.Compras, i).dicc, p).cantHackeable = +c$ ▷ $O(log(I) + O(log(P))$
49: **else**
50: $itLocal \leftarrow Significado(Significado(l.Items, i), p)$ ▷ $O(log(I) + O(log(P))$
51: $Definir(Significado(persona.Compras, i).dicc, p, < itLocal, c >$ ▷ $O(log(I) + O(log(P))$
52: **end if**
53: **end if**
54: **end if**

✓Complejidad: $O(log(P) + log(I) + log(cant) + log(A))$
✓Justificación: $log(P)+log(I)+log(cant)+log(A)+log(A)+log(A)+log(A)+log(A)+log(A)+log(I)+log(I)+log(P)+log(I)+log(P)$; Hay algunos casos en la funcion donde hay dos complejidas una en el la parte del if y otra en la del else, por tanto nunca se correran ambas complejidades, por ejemplo en la parte de dto=0 dentro tenemos en la seccion del if del primer definido $O(log(I)+log(P)+log(I))$ o la del else que es $O(log(I)+log(P)+log(I)+log(P)$. Por otro lado, la parte de gastos esta acotado por la cantidad de personas puesto que a lo sumo en peor caso todas las personas tiene gastos totales distintos consiguiendo asi A nodos

iPuestoConMenorStock(**in** l : `Lolla`, **in** i : `Item`) $\rightarrow$ res : IDPuesto

1: $itPuesto \leftarrow CrearIt(Significado(i, l.Items))$ ▷ $O(Log(I))$
2: $Actual \leftarrow itPuesto$
3: **while** $HaySiguiente?(itPuesto)$ **do** ▷ $O(P)$
4: $PunteroPuesto \leftarrow SiguienteSignificado(itPuesto)$ ▷ $\Theta(1)$
5: $ActualPuesto \leftarrow SiguienteSignificado(Actual)$ ▷ $\Theta(1)$
6: **if** $Stock(SiguienteSignificado(PunteroPuesto)) < Stock(SiguienteSignificado(ActualPuesto)$ **then** ▷ $\Theta(1)$
7: $Actual \leftarrow itPuesto$
8: **end if**
9: $Avanzar(itPuesto)$ ▷ $\Theta(1)$
10: **end while**
11: $res \leftarrow SiguienteClave(Actual)$ ▷ $\Theta(1)$

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

Justificación: El algoritmo hace una busqueda en un diccionario logaritmico que tiene como complejidad para la busuqeda en este O($log(I)$) y luego vamos a buscar en otro diccionario en el cual veremos todos los puestos lo que tiene complejidad de O(P) y no hay nada mas con una complejidad distinta de $\Theta(1)$

- Agregar comentarios descriptivos
- Elegir mejor nombre de variables

---

iHackear(**in/out** l : `Lolla`, **in** i : `Item`, **in** p : `DNI`)

1: $humano \leftarrow Significado(l.Personas, p)$ ▷ $O(log(A))$
2: $PuestoAHackear \leftarrow Significado(humano.Compras, i)$ ▷ $O(log(I))$
3: $puestoMinimo \leftarrow PuestoAHackear.puestoHack$ ▷ $\Theta(1)$
4: $SiguienteSignificado(puestoMinimo).CantHackeable \leftarrow SiguienteSignificado(puestoMinimo).CantHackeable - 1$ ▷ $\Theta(1)$
5: **if** $SiguienteSignificado(puestoMinimo).CantHackeable = 0$ **then** ▷ $\Theta(1)$
6: $Borrar(PuestoAHackear.Dicc, SiguienteClave(PuestoAHackear.PuestoHack))$ ▷ $O(log(P))$
7: $PuestoAHackear.PuestoHack \leftarrow CrearIt(PuestoAHackear.Dicc)$ ▷ $O(log(P))$
8: **end if**
9: $itemHackeado \leftarrow Significado(puestoMinimo.items, i)$ ▷ $O(log(I))$
10: $itemHackeado.stock \leftarrow itemHackeado.stock + 1$ ▷ $\Theta(1)$
11: $precioItem \leftarrow itemHackeado.precio$ ▷ $\Theta(1)$
12: $personaEnPuesto.gasto \leftarrow personaEnPuesto.gasto - precioItem$ ▷ $\Theta(1)$
13: **if** $personaEnPuesto = 0$ **then** ▷ $\Theta(1)$
14: $Borrar(puestoMinimo.clientes, p)$ ▷ $O(log(A))$
15: **end if**
16: $gastoPrevio \leftarrow humano.gastoTotal$ ▷ $\Theta(1)$
17: $humano.gastoTotal \leftarrow humano.gastoTotal - precioItem$ ▷ $\Theta(1)$
18: $nodoAnterior \leftarrow Significado(l.gastos, gastoPrevio)$ ▷ $O(log(A))$
19: **if** $Definido?(l.gastos, humano.gastoTotal)$ **then** ▷ $O(log(A))$
20: $Ag(p, Significado(l.gastos, humano.gastoTotal))$ ▷ $O(log(A))$
21: **else**
22: $Definir(l.gastos, humano.gastoTotal, Ag(p, vacio))$ ▷ $O(log(A))$
23: **end if**
24: **if** $p = maxGastador.id$ **then** ▷ $\Theta(1)$
25: **if** $\#(nodoAnterior) = 1$ **then** ▷ $\Theta(1)$
26: $it \leftarrow CrearIt(AnteriorSignificado(maxGastador.puntero))$ ▷ $O(log(A))$
27: $maxGastador.puntero \leftarrow Anterior(maxGastador.puntero)$ ▷ $\Theta(1)$
28: $maxGastador.id \leftarrow Siguiente(it)$ ▷ $\Theta(1)$
29: **else**
30: $it \leftarrow CrearIt(significado(l.gastos, gastoPrevio))$ ▷ $O(log(A))$
31: $Avanzar(it)$ ▷ $\Theta(1)$
32: $maxGastador.id \leftarrow Siguiente(it)$ ▷ $\Theta(1)$
33: **end if**
34: **end if**
35: **if** $\#(nodoAnterior) = 1$ **then** ▷ $\Theta(1)$
36: $Borrar(l.gastos, gastoPrevio)$ ▷ $O(log(A))$
37: **else**
38: $Eliminar(nodoAnterior, p)$ ▷ $O(log(A))$
39: **end if**

puestoMinimo es un iterador

Que es personaEnPuesto?

Complejidad: $O(log(A) + log(P) + log(I))$
Justificación: El algoritmo hace una busqueda en un diccionario logaritmico que tiene como complejidad para la busuqeda en este O($log(I)$) y luego vamos a buscar en una secuencia con todos los puestos lo que tiene complejidad de O(P) y no hay nada mas con una complejidad distinta de $\Theta(1)$

---

iPersonas(**in** l : `Lolla`) → res : $Conj(DNI)$

1: $res \leftarrow l.ConjDNIs$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$
Justificación: El algoritmo solamente saca de manera lineal el conjunto de personas guardado en el lolla $\Theta(1)$

---

iPuestos(**in** l : `Lolla`) → res : $diccLog(IDPuesto, puesto)$

1: $res \leftarrow l.puestos$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$
Justificación: El algoritmo solamente saca de manera lineal el dicc de puestos guardado en el lolla $\Theta(1)$

---

iGastoDePersona(**in** l : `Lolla`, **in** p : `DNI`) → res : nat

1: $res \leftarrow Significado(l.Personas, p).GastoTotal$ ▷ $O(log(A))$

Complejidad: $O(log(A))$
Justificación: El algoritmo hace una busqueda en un diccionario logaritmico en el cual la cantidad de elementos que tiene es A, por lo tanto la busuqeda es $O(log(A))$

---

iMaximoGastador(**in** l : `Lolla`) → res : DNI

1: $res \leftarrow l.MaximoGastador.id$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$
Justificación: El algoritmo solamente saca de manera lineal el id guardado en el lolla $\Theta(1)$

4.2 Axiomas Puesto

✓iStock(**in** p : `puesto`, **in** i : `Item`) $\rightarrow$ res : Nat

1: $res \leftarrow Significado(p.items, i).stock$ ▷ $O(log(I))$

Complejidad: $O(log(I))$
Justificación: El algoritmo cuenta con una asignacion a res del Significado de la clave i en el diccionario p, como el diccionario devuelve una tupa, indico que el valor que me interesa es stock. esta asignacion tiene costo $\Theta(1)$, pero la funcion Significado es $O(log(I))$

✓iCuantoGasto(**in** p : `puesto`, **in** per : `DNI`) $\rightarrow$ res : dinero

1: $res \leftarrow Significado(p.clientes, per)$ ▷ $O(log(A))$

Complejidad: $O(log(A))$
Justificación: El algoritmo cuenta con una asigacion a res del sigificado del diccionario per, como el diccionario p tiene varios elementos, devuelvo el valor almacenado en clientes. esta asignacion tiene costo $\Theta(1)$, pero la funcion Significado es $O(log(A))$

---

iNuevoPuesto(**in** p: `dicc(item, nat)`,**in** s: `dicc(item, nat)`, **in** d: `dicc(item, dicc(cant,nat))`) $\rightarrow res$: puesto

1: $I \leftarrow Vacio()$ ▷ $\Theta(1)$
2: $it \leftarrow CrearIt(p)$ ▷ $\Theta(1)$
3: **while** $HaySiguiente(it)$ **do** ▷ $O(I)$
4: $key \leftarrow SiguienteClave(it)$ ▷ $\Theta(1)$
5: **if** $Definido?(d, key)$ **then** ▷ $O(Log(I))$
6: $Definir(I, key, < Significado(p, key), Significado(s, key), Significado(d, key) >$) ▷ $O(Log(I))$
7: $Borrar(p, key)$ ▷ $O(Log(I))$
8: $Borrar(s, key)$ ▷ $O(Log(I))$
9: $Borrar(d, key)$ ▷ $O(Log(I))$
10: **else**
11: $Definir(I, key, < Significado(p, key), Significado(s, key) >)$ ▷ $O(Log(I))$
12: $Borrar(p, key)$ ▷ $O(Log(I))$
13: $Borrar(s, key)$ ▷ $O(Log(I))$
14: **end if**
15: $Avanzar(it)$ ▷ $\Theta(1)$
16: $EliminarAnterior(it)$ ▷ $\Theta(1)$
17: **end while**
18: $res \leftarrow < I, Vacio() >$

Complejidad: $O(I \times log(I)$
Justificación: La I viene de que hay que pasar todos los items uno por uno

iDescuentoDeItem(**in** p : `puesto`, **in** i : `item`, **in** $cant$: `nat`) $\rightarrow$ res : nat

Esto es una tupla

```
1:  if Definido?(Significado(p.items, i), cant) then   ▷ O(log(I) + log(cant))
2:      res ← Significado(Significado(p.items, i), cant)   ▷ O(log(I) + log(cant))
3:  else
4:      it ← Definir(Significado(p.items, i), cant, 0)   ▷ O(log(I) + log(cant))
5:      if HayAnterior(it) then   ▷ Θ(1)
6:          res ← AnteriorSignificado(it)   ▷ Θ(1)
7:          Borrar(Significado(p.items, i), cant)   ▷ O(log(I) + log(cant))
8:      else
9:          res ← 0   ▷ Θ(1)
10:     end if
11: end if
```

Complejidad: $O(log(I) + log(cant))$
Justificación: Para entrar al diccionario de descuentos de un item en especifico, se necesitan a lo sumo log(I) iteraciones, luego para ver si esta definido, o en casos posteriores definir y borrar, la cantidad adecuada en el diccionario se necesitan a lo sumo log(cant) iteraciones (esto es si hay un descuento definido por cada cantidad). Por ultimo, crear el iterador se hace en log(cant) iteraciones para encontrarlo, y se le suman log(I) para encontrar el item dentro del puesto.

5 Aclaraciones:

1. Tomamos que al crear un iterador, este se posicionara en el menor elemento (ya sea en un diccionario logaritmico, un conjunto, etc) y avanzara en orden ascendente

2. Si bien en la parte de ABS del Puesto no esta plasmado el observador Ventas, su accionar no lo utilizamos para las estructuras ni las funciones del modulo. Esto se debe a que no se pedia explicitamente en el enunciado cada compra con su gasto sino el total gastado por persona; a la hora de hackear, tan solo nos guardamos las compras que no tuvieron descuento para operar sobre ellas.