Algoritmos y Estructuras de Datos II

Trabajo Práctico 1

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

Lollapatuza

Grupo *Tere*

Integrante	LU	Correo electrónico
Tercic, Magalí	581/21	tercicmagali@gmail.com
Piñeiro, Ariel	128/22	arielalhuepd@gmail.com
Schandin, Juan	960/21	jschandin@gmail.com
Vega, Martina	596/22	martina.sandra.vega@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Contents

1 Comentarios

- Consideramos que una persona realiza un pedido todo al mismo tiempo, es decir, no ocurre que pide 2 hamburguesas, 1 gaseosa y, después de pensarlo mejor, pide 1 hamburguesa más. En todo caso, nosotros consideramos que esa compra son 3 hamburguesas y 1 gaseosa en total. Sí puede ocurrir que la misma persona compre el mismo item en dos compras seguidas, pero en un acto de compra particular de la persona agrupamos los items.
- Consideramos que las personas habilitadas a hacer compras son el conjunto de las mismas que "están" en el festival, reflejado en el generador del TAD correspondiente.
- Cada puesto ya decidió sus promociones al momento del nuevo festival (esto está reflejado en el generador del TAD "Puesto").
- Agregamos nombres a los puestos (un número) ya que nos parecía razonable que hubieran dos puestos con el mismo menú, stock, promociones y compras al momento de iniciar un festival (por ejemplo pensar en dos máquinas expendedoras iguales); para diferenciarlos, entonces, le asignamos nombres únicos a cada puesto del conjunto de puestos con que comienza el festival.
- Preguntar quién gastó más en un Lollapatuza sin gente nos parece que no tiene sentido ya que se nos pide devolver una persona, si no hay personas sería arbitrario devolver cualquier cosa.

2 Predicados auxiliares

// Se fija que si hay items compartidos entre los menús de distintos puestos, entonces tengan el mismo precio.

$$\begin{aligned}\text{preciosEstandar}(C_{puesto}) \equiv\ & (\forall p_1, p_2\text{: } puesto, \forall i\text{: Item})(\\ & (\\ & \quad \{p_1, p_2\} \subseteq C_{puesto} \wedge \\ & \quad i \in (items(menu(p_1)) \cap items(menu(p_2))) \\ &) \Rightarrow_{\text{L}} (\\ & \quad \text{precio(menu}(p_1), i) = \text{precio(menu}(p_2), i) \\ &) \\ &)\end{aligned}$$

// Se fija si un puesto admite hackeo, esto es, existe una compra sin descuento para una persona dada y un item dado en un puesto.

$$\begin{aligned}\text{puestoAdmiteHackeo}(pu, pe, i) \equiv\ & (\exists \text{c: Compra})(\\ & \quad \text{está?(c, compras}(pu, pe)) \wedge \\ & \quad \text{c[item]} = \text{i} \wedge \\ & \quad \neg \text{c[descuento]} \\ &)\end{aligned}$$

// Se fija si un festival admite hackeo, esto es, que la persona dada sea una persona habilitada, y que exista un puesto del festival que admita hackeo.

$$\begin{aligned}\text{festivalAdmiteHackeo}(l, pe, i) \equiv\ & (\exists \text{pu: Puesto})(\\ & \quad \text{pe} \in \text{personas}(l) \wedge \\ & \quad \text{pu} \in \text{puestos}(l) \wedge \\ & \quad \text{puestoAdmiteHackeo}(pu, pe, i) \\ &)\end{aligned}$$

// Se fija que no existan dos promociones para el mismo item para la misma cantidad en un conjunto de promociones.

$$\begin{aligned}\text{promosÚnicas}(PR) \equiv\ & (\forall pr_1, pr_2\text{: Promo})(\\ & \quad \{pr_1, pr_2\} \subseteq PR \Rightarrow_{\text{L}} pr_1[item] \neq pr_2[item] \vee pr_1[cantidad] \neq pr_2[cantidad] \\ &)\end{aligned}$$

// Se fija que no se repitan nombres en un conjunto de puestos.

$$\text{nombresÚnicos}(C_{puesto}) \equiv (\forall p_1, p_2\text{: Puesto})(\ \{p_1, p_2\} \subseteq C_{puesto}\ \Rightarrow\ p_1[nombre] \neq p_2[nombre]\ )$$

3 TADs

3.1 Extensiones y renombres

TAD String **ES** secu(Letra)

TAD Item **ES** String

TAD Nombre **ES** Nat

TAD Persona **ES** Nat

TAD Consumos **ES** conj(ConsumoSimple)

TAD ConsumoSimple **ES** Tupla(Item, Nat)

géneros ConsumoSimple

usa Item, Nat

otras operaciones

- [item] : ConsumoSimple $\longrightarrow$ String
- [cantidad] : ConsumoSimple $\longrightarrow$ Nat

axiomas $\forall c$: ConsumoSimple

c[item] $\equiv c_1$

c[cantidad] $\equiv c_2$

Fin TAD

TAD Compra **ES** Tupla(String, Nat, Nat, Bool, String)

géneros Compra

usa String, Nat, Bool

otras operaciones

- [item] : Compra $\longrightarrow$ String
- [cantidad] : Compra $\longrightarrow$ Nat
- [total] : Compra $\longrightarrow$ Nat
- [promo] : Compra $\longrightarrow$ Bool
- [persona] : Compra $\longrightarrow$ String

axiomas $\forall c$: Compra

c[item] $\equiv v_1$

c[cantidad] $\equiv v_2$

c[total] $\equiv v_3$

c[promo] $\equiv v_4$

c[persona] $\equiv v_5$

Fin TAD

TAD PROMO **ES** Tupla(String, Nat, Nat)

géneros Promo

usa String, Nat

otras operaciones

- [item] : Promo $\longrightarrow$ String
- [cantidad] : Promo $\longrightarrow$ Nat
- [descuento] : Promo $\longrightarrow$ Nat

axiomas $\forall p$: Promo

p[item] $\equiv p_1$

p[cantidad] $\equiv p_2$

p[descuento] $\equiv p_3$

Fin TAD

3.2 Lollapatuza

TAD LOLLAPATUZA

géneros Lollapatuza

usa Puesto, Persona, Consumos, conj, Nat

exporta maxGastador, observadores, generadores

igualdad observacional

$$(\forall l_1, l_2 : \text{Lollapatuza}) \left(l_1 =_{\text{obs}} l_2 \iff \begin{pmatrix} \text{puestos}(l_1) =_{\text{obs}} \text{puestos}(l_2) \wedge \\ \text{personas}(l_1) =_{\text{obs}} \text{personas}(l_2) \end{pmatrix} \right)$$

observadores básicos

puestos	: Lollapatuza	$\longrightarrow$ conj(Puesto)
personas	: Lollapatuza	$\longrightarrow$ conj(Persona)

generadores

nuevoFestival	: conj(Puesto) C_{puesto} × conj(Persona) $C_{persona}$	$\longrightarrow$ Lollapatuza $\{\text{preciosEstandar}(C_{puesto}) \wedge \neg\text{vacío?}(C_{persona}) \wedge \text{nombresÚnicos}(C_{puesto}\}$
comprarEnPuesto	: Lollapatuza l × Puesto pu × Persona pe × Consumos C	$\longrightarrow$ Lollapatuza $\{\text{pe} \in \text{personas}(l) \wedge \text{pu} \in \text{puestos}(l) \wedge \text{consumoPosible}(pu, C)\}$
hackearFestival	: Lollapatuza l × Persona pe × Item i	$\longrightarrow$ Lollpatuza $\{\text{pe} \in \text{personas}(l) \wedge \text{festivalAdmiteHackeo}(l, pe, i)\}$

otras operaciones

// Calcula lo que gastó una persona en todos los puestos en los que realizó compras.

totalPorPersona	: conj(Puesto) × Persona pe	$\longrightarrow$ Nat	$\{\text{pe} \in \text{personas}(l)\}$

// Devuelve la persona que más dinero gastó.

maxGastador	: Lollapalloza l	$\longrightarrow$ Persona $\{\neg\text{vacío?}(\text{personas}(l))\}$

// Auxiliares.

buscarMaxGastador	: conj(Puesto) × conj(Persona) P × Persona pe	$\longrightarrow$ Persona $\{\text{pe} \in \text{personas}(l) \wedge (\forall p\text{: Persona})(\text{p} \in \text{P} \Rightarrow \text{p} \in \text{personas}(l))\}$
comprarAux	: conj(Puesto) × Puesto × Persona × Consumos	$\longrightarrow$ conj(Puesto)
hackearAux	: conj(Puesto) × Persona × Item	$\longrightarrow$ conj(Puesto)

axiomas $\forall l$: Lollapatuza, $\forall pe, pe_1, pe_2$: Persona, $\forall C_{puesto}$: conj(Puestos), $\forall C_{persona}$: conj(Persona), $\forall C$: Consumo, $\forall i$: Item

// Puestos

puestos(nuevoFestival(C_{puesto}, $C_{persona}$))	$\equiv$ C_{puesto}
puestos(comprarEnPuesto(l, pu, pe, C))	$\equiv$ comprarAux(puestos(l), pu, pe, C)
puestos(hackearFestival(l, pe, i))	$\equiv$ hackearAux(puestos(l), pe, i)

// Personas

personas(nuevoFestival(C_{puesto}, $C_{persona}$))	$\equiv$ $C_{persona}$
personas(comprarEnPuesto(l, pu, pe, C))	$\equiv$ personas(l)

personas(hackearFestival(l, pe, i)) ≡ personas(l)

// Otras operaciones

totalPorPersona(C_{puesto}, pe) ≡ **if** vacío?(C_{puesto}) **then**
0
else
sumaTotalCompras(compras(dameUno(C_{puesto}), pe)) +
totalPorPersona(sinUno(C_{puesto}), pe)
fi

maxGastador(l) ≡ buscarMaxGastador(
puestos(l),
sinUno(personas(l)),
dameUno(personas(l))
)

buscarMaxGastador(C_{puesto}, $C_{persona}$, pe) ≡ **if** vacío?($C_{persona}$) **then**
pe
else
if totalPorPersona(C_{puesto}, dameUno($C_{persona}$)) ≥
totalPorPersona(C_{puesto}, pe) **then**
buscarMaxGastador(
C_{puesto},
sinUno($C_{persona}$),
dameUno($C_{persona}$)
)
else
buscarMaxGastador(C_{puesto}, sinUno($C_{persona}$), pe)
fi
fi

comprarAux(C_{puesto}, pu, pe, C) ≡ **if** vacío?(C_{puesto}) **then**
∅
else
if dameUno(C_{puesto}) = pu **then**
Ag(cobrar(pu, pe, C), sinUno(C_{puesto}))
else
Ag(
dameUno(C_{puesto}),
comprarAux(sinUno(C_{puesto}), pu, pe, C)
)
fi
fi

hackearAux(C_{puesto}, pe, i) ≡ **if** vacío?(C_{puesto}) **then**
∅
else
if hayCompraSinDescuento(
compras(dameUno(C_{puesto}), pe), i
) **then**
Ag(
hackearPuesto(dameUno(C_{puesto}), pe, i),
sinUno(C_{puesto})
)
else
Ag(
dameUno(C_{puesto}),
hackearAux(sinUno(C_{puesto}), pe, i)
)
fi
fi

Fin TAD

3.3 Puesto de comida

TAD PUESTO

géneros Puesto

exporta sumaTotalCompras, consumoPosible, hayCompraSinDescuento, generadores, observadores

usa ITEM, PERSONA, PROMO, CONSUMOS, COMPRA, STOCK, MENU, SECU, NAT, CONJ

igualdad observacional

$$(\forall p_1, p_2 : \text{Puesto}) \left(p_1 =_{\text{obs}} p_2 \iff \begin{pmatrix} \text{nombre}(p_1) =_{\text{obs}} \text{nombre}(p_2) \land \\ \text{menu}(p_1) =_{\text{obs}} \text{menu}(p_2) \land \\ \text{stock}(p_1) =_{\text{obs}} \text{stock}(p_2) \land \\ \text{promos}(p_1) =_{\text{obs}} \text{promos}(p_2) \land \\ (\forall pe\text{: Persona})(\text{compras}(p_1, pe) =_{\text{obs}} \text{compras}(p_2, pe)) \end{pmatrix} \right)$$

observadores básicos

nombre	: Puesto	$\longrightarrow$ Nombre
menu	: Puesto	$\longrightarrow$ Menu
stock	: Puesto	$\longrightarrow$ Stock
promos	: Puesto	$\longrightarrow$ conj($Promo$)
compras	: Puesto $\times$ Persona	$\longrightarrow$ secu($Compra$)

generadores

nuevoPuesto	: Nombre $\times$ Stock $\times$ Menu m $\times$ conj(Promo) PR {promosÚnicas(PR) $\land$ ($\forall p$: Promo)(p $\in$ PR $\Rightarrow$ (p[item] $\in$ items(m) $\land$ 0 < p[descuento] < 100)}	$\longrightarrow$ Puesto
cobrar	: Puesto pu $\times$ Persona $\times$ Consumos C	$\longrightarrow$ Puesto {consumoPosible(pu, C)}
hackearPuesto	: Puesto pu $\times$ Persona pe $\times$ Item i	$\longrightarrow$ Puesto {puestoAdmiteHackeo(pu, pe, i)}

otras operaciones

// Transforma un conjunto de consumos simples en una secuencia de compras.

procesarConsumos	: Puesto pu $\times$ Persona $\times$ Consumos C {($\forall cs$: ConsumoSimple)(cs $\in$ C $\Rightarrow$ cs[item] $\in$ items(menu(pu)))}	$\longrightarrow$ secu($Compra$)

// Calcula el gasto total para un consumo simple.

totalConsumoSimple	: Puesto pu $\times$ ConsumoSimple cs	$\longrightarrow$ Nat {cs[item] $\in$ items(menu(pu))}

// Se fija si una promo en particular es aplicable para un consumo simple.

aplicaPromo?	: Promo $\times$ ConsumoSimple	$\longrightarrow$ Bool

// Se fija si, dado un conjunto de promos, existe alguna en ese conjunto que aplique para el consumo simple.

aplicaAlMenosUnaPromo?	: conj(Promo) $\times$ ConsumoSimple	$\longrightarrow$ Bool

// Selecciona aquella promoción que mejor se aplica para la cantidad de items del consumo simple.

obtenerMejorDescuento	: conj(Promo) PR $\times$ ConsumoSimple cs	$\longrightarrow$ Nat

// Dada una secuencia de compras, devuelve la facturación total.

sumaTotalCompras	: secu($Compra$)	$\longrightarrow$ Nat

// Se fija si en una secuencia de compras existe una de ellas sin descuento para un item dado.

hayCompraSinDescuento	: secu(Compra) × Item i	$\longrightarrow$	Bool

// Se fija si, dado un conjunto de consumos simples, todos ellos se pueden realizar en un puesto determinado.

consumoPosible	: Puesto × Consumos	$\longrightarrow$	Bool

// Auxiliares.

disminuirStockAux	: Stock × Consumos	$\longrightarrow$	Stock
hackearPuestoAux	: secu(Compra) × Item	$\longrightarrow$	secu(Compra)
hackearCompra	: Puesto pu × Compra c	$\longrightarrow$	Compra {c[item] ∈ items(menu(pu))]}

// Enunciado

div	: Nat × Nat k	$\longrightarrow$ Nat	{0 < k}
aplicarDescuento	: Nat × Nat d	$\longrightarrow$ Nat	{d < 100}

axiomas $\forall pu$: Puesto, $\forall n$: Nombre, $\forall i$: Item, $\forall s$: Stock, $\forall pr$: Promo, $\forall pe, pe_1, pe_2$: Persona, $\forall m$: Menu, $\forall c$: Compra, $\forall C$: Consumos, $\forall cs$: ConsumoSimple, $\forall PR$: conj(Promo), $\forall sc$: secu($Compra$), $\forall n_1, n_2$: Nat

// Nombre

nombre(nuevoPuesto(n, s, m, PR))	≡	n
nombre(cobrar(pu, C, pe))	≡	nombre(pu)
nombre(hackearPuesto(pu, pe, i))	≡	nombre(pu)

// Menu

menu(nuevoPuesto(n, s, m, PR))	≡	m
menu(cobrar(pu, C, pe))	≡	menu(pu)
menu(hackearPuesto(pu, pe, i))	≡	menu(pu)

// Stock

stock(nuevoPuesto(n, s, m, PR))	≡	s
stock(cobrar(pu, C, pe))	≡	disminuirStockAux(stock(pu), C)
stock(hackearPuesto(pu, pe, i))	≡	hackearStock(stock(pu), i)

// Promos

promos(nuevoPuesto(n, s, m, PR))	≡	PR
promos(cobrar(pu, i, c, p_e))	≡	promos(pu)
promos(hackearPuesto(pu, pe, i))	≡	promos(pu)

// Compras

compras(nuevoPuesto(n, s, m, PR), pe)	≡	<>
compras(cobrar(pu, pe_1, C), pe_2)	≡	**if** $pe_1 =_{\text{obs}} pe_2$ **then** procesarConsumos(pu, pe_2, C) & compras(pu, pe_2) **else** compras(pu, pe_2) **fi**

compras(hackearPuesto(pu, pe, i)) ≡ hackearPuestoAux(compras(pu, pe), i)

// Otras operaciones

procesarConsumos(pu, pe, C) ≡ **if** vacío?(C) **then**
<>
else
Compra(
dameUno(C)[item],
dameUno(C)[cantidad],
totalConsumoSimple(pu, dameUno(C)),
aplicaAlMenosUnaPromo?(pu, dameUno(C)),
pe
) • procesarConsumos(pu, pe, sinUno(C))
fi

totalConsumoSimple(pu, cs) ≡ **if** aplicaAlMenosUnaPromo?(promos(pu), cs) **then**
aplicarDescuento(
precio(menu(pu), cs[$item$]) * cs[cantidad],
obtenerMejorDescuento(promos(pu), cs)
)
else
precio(menu(pu), cs[$item$]) * cs[cantidad]
fi

aplicaAlMenosUnaPromo?(PR, cs) ≡ **if** vacío?(PR) **then**
$false$
else
aplicaPromo?(dameUno(PR), cs) ∨
aplicaAlMenosUnaPromo?(sinUno(PR), cs)
fi

aplicaPromo?(pr, cs) ≡ pr[item] = cs[item] ∧ pr[cantidad] ≤ cs[cantidad]

obtenerMejorDescuento(PR, cs) ≡ **if** vacío?(PR) **then**
0
else
if aplicaPromo?(dameUno(PR), cs) **then**
max(
dameUno(PR)[descuento],
obtenerMejorDescuento(sinUno(PR), cs)
)
else
obtenerMejorDescuento(sinUno(PR), cs)
fi
fi

sumaTotalCompras(sc) ≡ **if** vacía?(sc) **then**
0
else
prim(sc)[total] + sumaTotalCompras(fin(sc))
fi

hayCompraSinDescuento(sc, i) ≡ **if** vacío?(sc) **then**
$false$
else
(prim(sc)[item] = i ∧ ¬prim(sc)[descuento]) ∨
hayCompraSinDescuento(fin(sc, i))
fi

consumoPosible(pu, C) ≡ **if** vacío?(C) **then**
$true$
else
(
dameUno(C)[item] ∈ items(stock(pu)) ∧
cantidad(stock(pu), dameUno(C)[item])
≥ dameUno(C)[cantidad]
) ∧ consumoPosible(pu, sinUno(C))
fi

hackearPuestoAux(sc, i) ≡ **if** vacío?(sc) **then**
∅
else
if prim(sc)[item] = i ∧ ¬prim(sc)[descuento] **then**
if prim(sc)[cantidad] - 1 = 0 **then**
// No queremos compras de 0 items
fin(sc)
else
// Modificamos la cantidad y el precio
hackearCompra(pu, prim(sc)) • fin(sc)
fi
else
prim(sc) • hackearPuestoAux(fin(sc), i)
fi
fi

hackearCompra(pu, c) ≡ Compra(
c[item],
c[cantidad] - 1,
precio(menu(pu), c[item]) * (c[cantidad] - 1),
c[descuento],
c[persona]
)

disminuirStockAux(s, C) ≡ **if** vacío?(C) **then**
s
else
disminuirStockAux(
disminuir(s, dameUno(C)[item], dameUno(C)[cantidad]),
sinUno(C)
)
fi

div(n_1, n_2) ≡ **if** $n_1 < n_2$ **then** 0 **else** $1 + div(n_1 - n_2, n_2)$ **fi**

aplicarDescuento(n_1, n_2) ≡ div($n_1 \times$ (100 - n_2), 100)

Fin TAD

3.4 Stock

TAD STOCK

géneros Stock

igualdad observacional

$$(\forall s_1, s_2 : \text{Stock}) \left(s_1 =_{\text{obs}} s_2 \iff \begin{pmatrix} \text{items}(s_1) =_{\text{obs}} \text{items}(s_2) \wedge_{\text{L}} \\ (\forall i\text{: Item})(\text{i} \in \text{items}(s_1) \Rightarrow_{\text{L}} \text{cantidad}(s_1, i) = \text{cantidad}(s_2, \\ i)) \end{pmatrix} \right)$$

exporta generadores, observadores

usa ITEM, NAT, DICC, CONJ

Stock

generadores

nuevo	: Dicc($Item$, Nat)	$\longrightarrow$	Stock	
disminuir	: Stock s $\times$ Item i $\times$ Nat n	$\longrightarrow$	Stock	{i $\in$ items(s) $\wedge_{\text{L}}$ cantidad(s, i) $\geq$ n}
hackearStock	: Stock s $\times$ Item i	$\longrightarrow$	Stock	{i $\in$ items(s)}

observadores básicos

items	: Stock	$\longrightarrow$	conj($Item$)
cantidad	: Stock s $\times$ Item i	$\longrightarrow$	Nat

axiomas $\forall d$: Dicc($Item$, Nat), $\forall s$: Stock, $\forall i, i_g, i_0$: Item, $\forall n$: Nat

items(nuevo(d))	$\equiv$	claves(d)
items(disminuir(s, i, n))	$\equiv$	items(s)
items(hackearStock(s, i))	$\equiv$	items(s)
cantidad(nuevo(d), i)	$\equiv$	**if** i $\in$ claves(d) **then** obtener(i, d) **else** 0 **fi**
cantidad(disminuir(s, i_g, n), i_o)	$\equiv$	**if** $i_g =_{\text{obs}} i_o$ **then** cantidad(s, i_o) - n **else** cantidad(s, i_o) **fi**
cantidad(hackearStock(s, i_g, n), i_o)	$\equiv$	**if** $i_g =_{\text{obs}} i_o$ **then** cantidad(s, i_o) + 1 **else** cantidad(s, i_o) **fi**

Fin TAD

3.5 Menu

TAD MENU

géneros Menu

igualdad observacional

$$(\forall m_1, m_2 : \text{Menu}) \left(m_1 =_{\text{obs}} m_2 \iff \begin{pmatrix} (items(m_1) =_{\text{obs}} items(m_2)) \wedge_{\text{L}} \\ (\forall i\text{: Item})(\text{i} \in \text{items}(m_1) \Rightarrow_{\text{L}} \text{precio}(m_1, i) = \\ \text{precio}(m_2, i)) \end{pmatrix} \right)$$

exporta generadores, observadores

usa ITEM, NAT, CONJ, DICC

observadores básicos

items	: Menu	$\longrightarrow$	conj($Item$)	
precio	: Menu m $\times$ Item i	$\longrightarrow$	Nat	$\{i \in \text{items}(m)\}$

generadores

nuevoMenu	: dicc($Item$, Nat)	$\longrightarrow$	menu

axiomas $\forall d$: dicc($Item$, Nat), $\forall i$: $Item$

items(nuevoMenu(d))	$\equiv$	claves(d)
precio(nuevoMenu(d), i)	$\equiv$	obtener(i, d)

Fin TAD