

# Algoritmos y Estructuras de Datos II

## Trabajo Práctico 2

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

**Lollapatuza**

TADgrupo

| Integrante             | LU     | Correo electrónico          |
|------------------------|--------|-----------------------------|
| Aquilante, Juan Pablo  | 755/18 | aquilantejp@outlook.es      |
| Bocanegra, Oscar Iván  | 537/22 | oscarbocanegraush@gmail.com |
| Flores Galvan, Silvina | 689/22 | silu204@gmail.com           |
| Sanchez, Gastón        | 361/22 | gasanchez@dc.uba.ar         |

**Reservado para la cátedra**

| Instancia       | Docente | Nota |
|-----------------|---------|------|
| Primera entrega |         |      |
| Segunda entrega |         |      |

# Índice

|  |           |
|--|-----------|
| <b>1. Módulo Lollapatuza</b>   | <b>3</b>  |
| 1.1. Interfaz . . . . .  | 3         |
| 1.2. Operaciones básicas de Lollapatuza . . . . .                                    | 3         |
| 1.3. Representación del lollapatuza . . . . .  | 4         |
| 1.4. Algoritmos . . . . .  | 7         |
| <b>2. Módulo PuestoDeComida</b>  | <b>10</b> |
| 2.1. Interfaz del módulo . . . . .   | 10        |
| 2.2. Operaciones básicas de Puesto . . . . .   | 10        |
| 2.3. Representación de puesto . . . . .  | 11        |
| 2.4. Algoritmos . . . . .  | 13        |
| <b>3. Módulo Cola de Prioridad Acotada(<math>\langle\alpha, \beta\rangle</math>)</b> | <b>15</b> |
| 3.1. TAD COLA DE PRIORIDAD ACOTADA( $\gamma$ ) . . . . .                             | 15        |
| 3.2. Interfaz . . . . .  | 16        |
| 3.3. Representación . . . . .  | 17        |
| 3.4. Algoritmos . . . . .  | 18        |
| 3.5. Funciones auxiliares . . . . .  | 19        |

# 1. Módulo Lollapatuza

## 1.1. Interfaz

### Interfaz

parámetros formales

géneros lolla

**función**  $\bullet = \bullet(\text{in } k_1 : \kappa, \text{in } k_2 : \kappa) \rightarrow res : \text{bool}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} (k_1 = k_2)\}$   
**Complejidad:**  $\Theta(\text{equal}(k_1, k_2))$   
**Descripción:** función de igualdad de  $\kappa$ 's

**función**  $\text{COPIAR}(\text{in } k : \kappa) \rightarrow res : \kappa$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} k\}$   
**Complejidad:**  $\Theta(\text{copy}(k))$   
**Descripción:** función de copia de  $\kappa$ 's

se explica con: LOLLAPATUZA

géneros: lolla

## 1.2. Operaciones básicas de Lollapatuza

**CREARLOLLA**(**in**  $ps : \text{dicc}(\text{puestoID}, \text{puesto})$ , **in**  $as : \text{conj}(\text{persona})$ )  $\rightarrow res : \text{lolla}$   
**Pre**  $\equiv \{\text{vendenAlMismoPrecio}(\text{significados}(ps)) \wedge \text{NoVendieronAun}(\text{significados}(ps)) \wedge \neg \emptyset?(as) \wedge \neg \emptyset?(\text{claves}(ps))\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{crearLolla}(ps, as)\}$   
**Complejidad:**  $\mathcal{O}(A * \log(A))$   
**Descripción:** genera un nuevo Lollapatuza.  
**Aliasing:** Se devuelve una referencia modificable

**VENDER**(**in/out**  $l : \text{lolla}$ , **in**  $pi : \text{puestoID}$ , **in**  $a : \text{persona}$ , **in**  $i : \text{item}$ , **in**  $c : \text{cant}$ )  
**Pre**  $\equiv \{l =_{\text{obs}} l_0 \wedge pi \in \text{claves}(l) \wedge a \in \text{personas}(l) \wedge \text{haySuficiente?}(\text{obtener}(pi, \text{puestos}(l)), i, c)\}$   
**Post**  $\equiv \{l =_{\text{obs}} \text{vender}(l_0, pi, a, i, c)\}$   
**Descripción:** vende una cantidad de un item determinado de un puesto del Lollapatuza.

**HACKEAR**(**in/out**  $l : \text{lolla}$ , **in**  $a : \text{persona}$ , **in**  $i : \text{item}$ )  
**Pre**  $\equiv \{l =_{\text{obs}} l_0 \wedge \text{ConsumioSinPromoEnAlgunPuesto}(l, a, i)\}$   
**Post**  $\equiv \{l =_{\text{obs}} \text{hackear}(l_0, a, i)\}$   
**Complejidad:**  $\mathcal{O}(\log(A) + \log(P) + \log(I))$   
**Descripción:** elimina el consumo del item pasado por parametro a la persona determinada, de alguna compra en la que no se le haya aplicado un descuento a este item. De existir el consumo en múltiples puestos de comida, se eliminará el mismo de alguno de ellos, viéndose afectado su Stock.

**PUESTOS**(**in**  $l : \text{lolla}$ )  $\rightarrow res : \text{dicc}(\text{puestoid}, \text{puesto})$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{puestos}(l)\}$   
**Complejidad:**  $\mathcal{O}(1)$   
**Descripción:** Devuelve los puestos de comida con sus IDs del Lollapatuza.  
**Aliasing:** Se devuelve una referencia modificable

**PERSONAS**(**in**  $l : \text{lolla}$ )  $\rightarrow res : \text{conj}(\text{persona})$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{personas}(l)\}$   
**Complejidad:**  $\mathcal{O}(1)$   
**Descripción:** devuelve el conjunto de personas en el Lollapatuza  
**Aliasing:** se devuelve el res por referencia

**Aliasing:** se devuelve res por referencia

**Descripción:** devuelve lo que gasto la persona en el lollapatuza

**Descripción:** devuelve lo que gasto la persona en el lollapatuza.

(8)  $\equiv$  Hay al menos algún puesto.  
 $\neg(\emptyset(\text{claves}(e.\text{puestos})))$

(9)  $\equiv$  En puestos no hay repetidos.  
 $(\forall p1, p2: \text{puesto})(\exists ID1, ID2 : \text{puestoID})(ID1 \neq ID2 \wedge \{ID1, ID2\} \subseteq \text{claves}(e.\text{puestos}) \wedge_L \text{obtener}(ID1, e.\text{puestos}) = p1 \wedge \text{obtener}(ID2, e.\text{puestos}) = p2) \Rightarrow_L p1 \neq p2$

(10)  $\equiv$  Todos los puestos venden el item al mismo precio.  
 $\text{vendenAlMismoPrecio}(\text{significados}(e.\text{puestos}))$

(11)  $\equiv$  Todas las personas de hackeables son válidas.  
 $\text{claves}(e.\text{hackeables}) \subseteq e.\text{personas}$

(12)  $\equiv$  Todas los puestosIDs que estan en hackeables son IDs de los puestos del lolla, y los punteros de hackeables apuntan a sus puestos correspondientes. Y todas las personas de hackeables hicieron una compra sin descuento  
 $(\forall per: \text{persona})(per \in \text{claves}(e.\text{hackeables}) \Rightarrow_L$   
 $(\forall i: \text{item})(i \in \text{claves}(\text{obtener}(per, e.\text{hackeables})) \Rightarrow_L$   
 $(\forall ID: \text{puestoID})(ID \in \text{claves}(\text{obtener}(i, \text{obtener}(per, e.\text{hackeables}))) \wedge ID \in \text{claves}(e.\text{puestos}) \Rightarrow_L$   
 $(\exists pe: \text{puntero}(\text{puesto}))(pe = \text{obtener}(ID, \text{obtener}(i, \text{obtener}(per, e.\text{hackeables}))) \wedge \text{def?}(ID, e.\text{puestos}) \wedge_L \text{obtener}(ID, e.\text{puestos}) = *pe))))))$

(13)  $\equiv$  Todas las personas en hackeables hicieron una compra sin descuento del item en el puesto.  
 $(\forall per: \text{persona})(per \in \text{claves}(e.\text{hackeables}) \Rightarrow_L$   
 $(\forall i: \text{item})(i \in \text{claves}(\text{obtener}(per, e.\text{hackeables})) \Rightarrow_L$   
 $(\forall ID: \text{puestoID})(ID \in \text{claves}(\text{obtener}(i, \text{obtener}(per, e.\text{hackeables}))) \Rightarrow_L$   
 $(\forall pe: \text{puntero}(\text{puesto}))(pe = \text{obtener}(ID, \text{obtener}(i, \text{obtener}(per, e.\text{hackeables}))) \Rightarrow_L$   
 $\neg(\emptyset = \text{ventasSinPromo}(*pe, per, i)))$

```

cantVendida : conj(puestos)  $\times$  conj(persona)  $\times$  item  $\longrightarrow$  cant
cantVendida(cp,cper,i)  $\equiv$  if vacio?(cp) then
    0
else
    cantVendidaPuesto(dameUno(cp),cper,i) + cantVendida(sinUno(cp),cper,i)
fi
cantVendidaPuesto : puesto  $\times$  conj(persona)  $\times$  item  $\longrightarrow$  cant
cantVendidaPuesto(p,cper,i)  $\equiv$  if  $\emptyset?$ (cper) then
    0
else
    cantVendidaPersona(ventas(p,dameUno(cper)),i) +
    cantVendidaPuesto(p,sinUno(cper),i)
fi
cantVendidaPersona : multiconj( $\langle$ item  $\times$  cant $\rangle$ )  $\times$  item  $\longrightarrow$  cant
cantVendidaPersona(vp,i)  $\equiv$  if vacia?(vp) then
    0
else
    if  $\pi_1$ (dameUno(vp)) = i then
         $\pi_2$ (dameUno(vp)) + cantVendidaPersona(sinUno(vp),i)
    else
        cantVendidaPersona(fin(vp),i)
    fi
fi
personasDeGastos : colaPriorA( $\langle$ dinero  $\times$  persona $\rangle$ )  $\longrightarrow$  conj(persona)
personasDeGastos(cPA)  $\equiv$  if vacia?(cPA) then
     $\emptyset$ 
else
    Ag( $\pi_2$ (proximo(cPA)), personasDeGastos(desencolar(cPA)))
fi
stockTotal : conj(puestos)  $\times$  item  $\longrightarrow$  cant
stockTotal(cp,i)  $\equiv$  if vacio?(cp) then 0 else stock(dameUno(cp),i) + stockTotal(sinUno(cp),i) fi

```

```

itemsLolla : conj(puesto) → conj(item)
itemsLolla(cp) ≡ if vacio?(cp) then ∅ else menu(dameUno(cp)) ∪ itemsLolla(sinUno(cp)) fi
sumaDeGastos : colaPriorA((dinero × persona)) → nat
sumaDeGastos(c) ≡ if vacia?(c) then 0 else π1(proximo(c)) + sumaDeGastos(desencolar(c)) fi
gastoMaximoPosible : dicc(puestoID × puesto) → nat
gastoMaximoPosible(d) ≡ if vacia?(claves(d)) then
    0
else
    gMaximoPosiblePuesto(obtener(dameUno(claves(d)))), menu(obtener(dameUno(claves(d))))
    + gastoMaximoPosible(borrar(dameUno(claves(d))), d)
fi
gMaximoPosiblePuesto : puesto × conj(item) → nat
gMaximoPosiblePuesto(p, m) ≡ if vacio?(m) then
    0
else
    precio(p, dameUno(m)) × stock(p, dameUno(m)) +
    gMaximoPosiblePuesto(p, sinUno(m))
fi
long : colaPriorA(α) → nat
long(c) ≡ if vacio?(c) then 0 else 1 + long(desencolar(c)) fi
•[•] : colaPriorA(α) → α
c[i] ≡ if i = 0 then proximo(c) else desencolar(c)[i-1] fi
gastoPersona : per × diccLog(puestoID × puesto) →
gastoPersona(per, d) ≡ sumaPorPuesto(per, significados(d))
sumaPorPuesto : per × conj(puesto) → dinero
sumaPorPuesto(per, c) ≡ if #c = 1 then
    gastoEnPuesto(ventas(dameUno(c), per), dameUno(c))
else
    gastoEnPuesto(ventas(dameUno(c), per), dameUno(c)) + sumaPorPuesto(c)
fi
gastoEnPuesto : multiconj((item × cant)) × puesto → dinero
gastoEnPuesto(m, p) ≡ if ∅?(m) then
    0
else
    aplicarDescuento(precio(p, π1(dameUno(m))), descuento(p, π1(dameUno(m)),
    π2(dameUno(m)))) × π2(dameUno(m)) + gastoEnPuesto(sinUno(m), p)
fi
Abs : estr e → lolla
Abs(e) ≡ puestos(l) =obs e.puestos ∧ personas(l) =obs e.personas

```

{Rep(e)}

## 1.4. Algoritmos

### Algoritmos

---

**iCrearLolla**(in  $ps$ : dicc(puestoID, puesto), in  $as$ : conj(persona))  $\rightarrow res$ : lollapatuza

```

1: diccGastos  $\leftarrow$  Vacio()  $\triangleright \mathcal{O}(1)$ 
2: cola  $\leftarrow$  Vacio()  $\triangleright \mathcal{O}(1)$ 
3: it  $\leftarrow$  crearIt(as)  $\triangleright \mathcal{O}(1)$ 
4: while HaySiguiente(it) do  $\triangleright \mathcal{O}(A)$ 
5:   puntero  $\leftarrow$  Encolar(cola, ( 0, Siguiente(it) ))  $\triangleright \mathcal{O}(\log(A))$ 
6:   Definir(diccGastos, Siguiente(it), puntero)  $\triangleright \mathcal{O}(\log(A))$ 
7:   Avanzar(it)  $\triangleright \mathcal{O}(1)$ 
8: end while
9: res  $\leftarrow$  ( as, diccGastos, cola, ps, Vacio() )  $\triangleright \mathcal{O}(1)$ 

```

Complejidad:  $\mathcal{O}(A * \log(A))$

Justificación: El while se realiza A veces, y por cada bucle se define una nueva clave en el diccionario diccGastos en  $\log(A)$  y se encola la cola que tiene en su significado, también en  $\log(A)$ .

---



---

**iVender**(in/out  $l$ : lolla, in  $pid$ : puestoID, in  $a$ : persona, in  $i$ : item, in  $c$ : cant)

```

1: //Accedo al puesto donde se va a realizar la venta
2: p  $\leftarrow$  Significado(l.puestos, pid)  $\triangleright \mathcal{O}(\log(P))$ 
3: precioItem  $\leftarrow$  Significado(p.menu, i)  $\triangleright \mathcal{O}(\log(I))$ 
4: //Defino el descuento,
5: //si no tiene descuento el array devuelve 0
6: if Definido?(p.descuentos,i) then  $\triangleright \mathcal{O}(\log(I))$ 
7:   descuento  $\leftarrow$  Significado(p.descuentos,i)[cant]  $\triangleright \mathcal{O}(\log(I))$ 
8: else
9:   descuento  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
10: end if
11: //Registro la venta en el puesto
12: Vender(p, a, i, c)  $\triangleright \mathcal{O}(\log(I) + \log(A))$ 
13: gastoVenta  $\leftarrow$  cant  $\times$  precioItem  $\times$  (100-descuento)/100  $\triangleright \mathcal{O}(1)$ 
14: //Si fue sin descuento y el puesto no era hackeable, añadirlo a hackeables
15: if descuento = 0  $\wedge$ 
16:    $\neg$  Definido?(Significado(Significado(l.hackeables, a), i), pid) then  $\triangleright \mathcal{O}(\log(A) + \log(I) + \log(P))$ 
17:   //creo un puntero y lo defino en l.hackeables
18:   punteroP  $\leftarrow$  &p
19:   Definir(Significado(Significado(l.hackeables,a),i),pid,punteroP)  $\triangleright \mathcal{O}(\log(A) + \log(I) + \log(P))$ 
20: end if
21: //Actualizo el gasto total de la persona en el lollapatuza
22: punteroAGasto  $\leftarrow$  Significado(l.punterosAGastos,a)  $\triangleright \mathcal{O}(\log(A))$ 
23: gastoActualizado  $\leftarrow$  *punteroAGasto + gastoVenta
24: gastoPer  $\leftarrow$  <gastoActualizado,a>  $\triangleright \mathcal{O}(1)$ 
25: punteroDinero  $\leftarrow$  Encolar(l.gastosPersona,gastoPer)  $\triangleright \mathcal{O}(\log(A))$ 
26: //Actualizo el puntero del gasto de la persona
27: Definir(l.punterosAGastos,a,punteroDinero)  $\triangleright \mathcal{O}(\log(A))$ 

```

Complejidad:  $\mathcal{O}(\log(A) + \log(P) + \log(I))$

---



---

**iPuestos**(in  $l$ : lollapatuza)  $\rightarrow res$ : diccLog(puestoID, puestos)

```

1: res  $\leftarrow$  l.puestos
   Complejidad:  $\mathcal{O}(1)$   $\triangleright \mathcal{O}(1)$ 

```

---

---



---

**iPersonas**(in  $l$ : lollapat, in  $k$ :  $\kappa$ )  $\rightarrow res$ :  $conj(persona)$

1:  $res \leftarrow l.personas$   
Complejidad:  $\mathcal{O}(1)$

---

$\triangleright \mathcal{O}(1)$

---



---

**iHackear**(in/out  $l$ : lollla, in  $a$ : persona, in  $i$ : item)

1:  $itP \leftarrow \text{CrearIt}(\text{Significado}(\text{Significado}(l.hackeables, a), \text{item}))$   $\triangleright \mathcal{O}(\log(A) + \log(I))$   
2:  $p \leftarrow *SiguienteSignificado(itP)$   $\triangleright \mathcal{O}(1)$   
3:  $listaVentas \leftarrow \text{Significado}(\text{Significado}(p.ventasSinDesc, a), i)$   $\triangleright \mathcal{O}(\log(A) + \log(I))$   
4:  $itLista \leftarrow \text{CrearIt}(listaVentas)$   $\triangleright \mathcal{O}(1)$   
5:  $it \leftarrow \text{Primero}(listaVentas)$   $\triangleright \mathcal{O}(1)$   
6: **if**  $\neg Siguiente(it).cant = 1$  **then**  $\triangleright \mathcal{O}(1)$   
7:      $Siguiente(it).cant \leftarrow (Siguiente(it).cant - 1)$   $\triangleright \mathcal{O}(1)$   
8: **else**  
9:      $\text{EliminarSiguiente}(it)$   $\triangleright \mathcal{O}(1)$   
10:     **if**  $\text{Longitud}(listaVentas) = 1$  **then** //Si el puesto deja de ser hackeable, cuesta  $\mathcal{O}(\log(P))$  borrarlo del diccionario  
11:          $\text{Definir}(\text{Significado}(p.ventasSinDesc, a), i, \text{Vacía}())$   $\triangleright \mathcal{O}(\log(A) + \log(I))$   
12:          $\text{Borrar}(\text{Significado}(\text{Significado}(l.hackeables, a), i), \text{SiguienteClave}(itP))$   $\triangleright \mathcal{O}(\log(P) + \log(A) + \log(I))$   
13:     **end if**  
14:      $\text{EliminarSiguiente}(itLista)$   $\triangleright \mathcal{O}(1)$   
15: **end if**  
16:  $\text{Definir}(p.stock, i, \text{Significado}(p.stock, i) + 1)$   $\triangleright \mathcal{O}(\log(I))$   
17:  $\text{precioItem} \leftarrow \text{Significado}(p.menu, i)$   $\triangleright \mathcal{O}(\log(I))$   
18:  $\text{Definir}(p.gastosDe, a, \text{Significado}(p.gastosDe, a) - \text{precioItem})$   $\triangleright \mathcal{O}(\log(A))$   
19:  $\text{puntero} \leftarrow \text{Significado}(l.punterosAGastos, a)$   $\triangleright \mathcal{O}(\log(A))$   
20:  $\text{gastoAnterior} \leftarrow *puntero$   $\triangleright \mathcal{O}(1)$   
21:  $\text{puntGastoAct} \leftarrow \text{Encolar}(l.gastosPersona, \langle \text{gastoAnterior} - \text{precioItem}, a \rangle)$   $\triangleright \mathcal{O}(\log(A))$   
22:  $\text{Definir}(l.punterosAGastos, a, \text{puntGastoAct})$   $\triangleright \mathcal{O}(\log(A))$   
Complejidad:  $\mathcal{O}(\log(A) + \log(P) + \log(I))$   
Justificación: El iterador del diccionario recorre inorden. Por lo tanto en linea 1 el it apunta al puesto de menor ID.

---



---



---

**iGastoTotal**(in  $l$ : lollla, in  $a$ : persona)  $\rightarrow res$ :  $nat$

1:  $\text{gasto} \leftarrow *(\text{Significado}(l.punterosAGastos, a))$   
2:  $res \leftarrow \text{gasto}$   
Complejidad:  $\mathcal{O}(\log(A))$

---

$\triangleright \mathcal{O}(\log(A))$   
 $\triangleright \mathcal{O}(1)$

---



---

**iQuienGastoMas**(in  $l$ : lollla)  $\rightarrow res$ : persona

$res \leftarrow (\text{Proximo}(l.gastosPersona)).per$   
Complejidad:  $\mathcal{O}(1)$

---

$\triangleright \mathcal{O}(1)$



---

```

iMenorStock(in l: lolla, in i: item) → res: puestoID
1: itP ← CrearIt(l.puestos)                                ▷  $\mathcal{O}(1)$ 
2: if Definido?(SiguienteSignificado(itP).stock, i) then    ▷  $\mathcal{O}(\log(I))$ 
3:   minStock ← Significado(SiguienteSignificado(itP).stock, i) ▷  $\mathcal{O}(\log(I))$ 
4: else
5:   minStock ← 0                                           ▷  $\mathcal{O}(1)$ 
6: end if
7: IDminStock ← SiguienteClave(itP)                        ▷  $\mathcal{O}(1)$ 
8: while HaySiguiente(itP) do                               ▷  $\mathcal{O}(P)$ 
9:   if Definido?(SiguienteSignificado(itP).stock, i) then ▷  $\mathcal{O}(\log(I))$ 
10:    stockActual ← Significado(SiguienteSignificado(itP).stock, i) ▷  $\mathcal{O}(\log(I))$ 
11:    IDstockActual ← SiguienteClave(itP)                  ▷  $\mathcal{O}(1)$ 
12:    if stockActual < minStock ∨ (stockActual = minStock ∧ IDstockActual < IDminStock) then ▷  $\mathcal{O}(1)$ 
13:      minStock ← stockActual                             ▷  $\mathcal{O}(1)$ 
14:      IDminStock ← IDstockActual                         ▷  $\mathcal{O}(1)$ 
15:    end if
16:  else
17:    IDstockActual ← SiguienteClave(itP)                  ▷  $\mathcal{O}(1)$ 
18:    if ¬(minStock = 0) then
19:      minStock ← 0                                         ▷  $\mathcal{O}(1)$ 
20:    end if
21:    if IDstockActual < IDminStock then
22:      IDminStock ← IDstockActual                         ▷  $\mathcal{O}(1)$ 
23:    end if
24:    Avanzar(itP)                                          ▷  $\mathcal{O}(1)$ 
25:  end if
26: end while
27: res ← IDminStock                                       ▷  $\mathcal{O}(1)$ 
   Complejidad:  $\mathcal{O}(P * \log(I))$ 

```

---

## 2. Módulo PuestoDeComida

### Interfaz

#### 2.1. Interfaz del módulo

parámetros formales

géneros puesto

función  $\bullet = \bullet(\text{in } k_1 : \kappa, \text{in } k_2 : \kappa) \rightarrow res : \text{bool}$   
Pre  $\equiv \{\text{true}\}$   
Post  $\equiv \{res =_{\text{obs}} (k_1 = k_2)\}$   
Complejidad:  $\Theta(\text{equal}(k_1, k_2))$   
Descripción: función de igualdad de  $\kappa$ 's

función  $\text{COPIAR}(\text{in } k : \kappa) \rightarrow res : \kappa$   
Pre  $\equiv \{\text{true}\}$   
Post  $\equiv \{res =_{\text{obs}} k\}$   
Complejidad:  $\Theta(\text{copy}(k))$   
Descripción: función de copia de  $\kappa$ 's

se explica con: PUESTODECOMIDA

géneros: puesto

#### 2.2. Operaciones básicas de Puesto

$\text{CREARPUESTO}(\text{in } p : \text{dicc}(\text{item}, \text{nat}), \text{in } s : \text{dicc}(\text{item}, \text{nat}), \text{in } d : \text{dicc}(\text{item}, \text{dicc}(\text{nat}, \text{nat})) \rightarrow res : \text{puesto}$   
Pre  $\equiv \{\text{claves}(p) = \text{claves}(s) \wedge \text{claves}(p) \subseteq \text{claves}(d)\}$   
Post  $\equiv \{res =_{\text{obs}} \text{crearPuesto}(m, s, d)\}$   
Complejidad:  $\mathcal{O}(I + \text{stockInicial} + \log(\text{cant}))$   
Descripción: genera un nuevo puesto.  
Aliasing: res se devuelve como una referencia modificable

$\text{VENDER}(\text{in/out } p : \text{puesto}, \text{in } a : \text{persona}, \text{in } i : \text{item}, \text{in } c : \text{cant})$   
Pre  $\equiv \{p_0 = p \wedge \text{haySuficiente?}(p, i, c)\}$   
Post  $\equiv \{p =_{\text{obs}} \text{vender}(p, a, i, c)\}$   
Complejidad:  $\mathcal{O}(\log(A) + \log(I))$   
Descripción: vende una cantidad de un item dado a una persona dada.

$\text{MENU}(\text{in } p : \text{puesto}) \rightarrow res : \text{conj}(\text{item})$   
Pre  $\equiv \{\text{true}\}$   
Post  $\equiv \{res =_{\text{obs}} \text{menu}(p)\}$   
Complejidad:  $\mathcal{O}(I)$   
Descripción: devuelve el menú de un puesto.  
Aliasing: res se devuelve como una referencia no modificable

$\text{PRECIO}(\text{in } p : \text{puesto}, \text{in } i : \text{item}) \rightarrow res : \text{dinero}$   
Pre  $\equiv \{i \in \text{menu}(p)\}$   
Post  $\equiv \{res =_{\text{obs}} \text{precio}(p, i)\}$   
Complejidad:  $\mathcal{O}(\log(I))$   
Descripción: devuelve el precio del item en el menu del puesto.

$\text{STOCK}(\text{in } p : \text{puesto}, \text{in } i : \text{item}) \rightarrow res : \text{cant}$   
Pre  $\equiv \{i \in \text{menu}(p)\}$   
Post  $\equiv \{res =_{\text{obs}} \text{stock}(p, i)\}$   
Complejidad:  $\mathcal{O}(\log(I))$   
Descripción: devuelve el stock de un item.

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

**Pre**  $\equiv \{i \in \text{menu}(p)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{descuento}(p, i, c)\}$

**Complejidad:**  $\mathcal{O}(\log(I))$

**Descripción:** devuelve el descuento correspondiente al puesto, item y cantidad comprados.

GASTOSDE(**in**  $p$ : puesto, **in**  $a$ : persona)  $\rightarrow res$  : dinero

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{gastosDe}(p, a)\}$

**Complejidad:**  $\mathcal{O}(\log(A))$

**Descripción:** devuelve el gasto realizado de la persona en el puesto

## Representación

### 2.3. Representación de puesto

puesto se representa con estr

donde estr es tupla( $menu$ : diccLog(item, precio)  
,  $stock$ : diccLog(item, cant)  
,  $gastosDe$ : diccLog(persona, nat)  
,  $descuentos$ : diccLog(item, arreglo\_dimensionable de nat)  
,  $ventasSinDesc$ : diccLog(persona, diccLog(item, lista(itLista)))  
,  $ventas$ : diccLog(persona, lista((item: item, cant: cant)))  
)

listaAMulticonj : secu( $\langle \text{item} \times \text{cant} \rangle$ )  $\rightarrow$  multiconj(item, cant)

listaAMulticonj( $l$ )  $\equiv$  **if** vacia?( $l$ ) **then**  $\emptyset$  **else** Ag(prim( $l$ ), listaAMulticonj(fin( $l$ ))) **fi**

$\bullet[\bullet]$  : secu( $\alpha$ )  $\times$  nat  $\rightarrow \alpha$

$s[i]$   $\equiv$  **if**  $i = 0$  **then** prim( $s$ ) **else** fin( $s$ )[ $i - 1$ ] **fi**

Rep : puesto  $\rightarrow$  bool

Rep( $l$ )  $\equiv$  true  $\iff$

(1)  $\wedge$  (2)  $\wedge$  (3)  $\wedge$  (4)  $\wedge_L$  (5)  $\wedge$  (6)  $\wedge$  (7)  $\wedge$  (8)  $\wedge$  (9)

(1)  $\equiv$  Los items que se venden en el menu deben ser los items del stock del puesto.

claves(e.menu) = claves(e.stock)

(2)  $\equiv$  Los items que tienen descuento estan contenidos en los items que se venden en el menú.

claves(e.descuentos)  $\subseteq$  claves(e.menu)

(3)  $\equiv$  Los items que aparecen en las ventas del puesto deben estar contenidos en el menú del puesto.

$(\forall p: \text{persona})(p \in \text{claves}(e.\text{ventasSinDesc}) \Rightarrow_L$

$((\forall i: \text{item})((\exists n: \text{nat})(0 \leq n < \text{long}(\text{obtener}(p, e.\text{ventas}())) \wedge_L \pi_1(\text{obtener}(p, e.\text{ventas}())[n]) = i) \Rightarrow_L$

$i \in \text{claves}(e.\text{menu})))$

(4)  $\equiv$  Las personas en la lista de gastos del puesto son las mismas que las personas en las ventas del puesto.

claves(e.ventas) = claves(e.gastosDe)

(5)  $\equiv$  El gasto de una persona debe coincidir con el costo de las compras de la misma en el puesto, considerando el precio, stock y descuento de items comprados.

$(\forall p: \text{persona})(p \in \text{claves}(e.\text{gastosDe}) \Rightarrow_L$

$\text{obtener}(p, e.\text{gastosDe}) = \text{gastoPuestoPersona}(\text{obtener}(p, e.\text{ventas}), e.\text{menu}, e.\text{descuentos}))$

(6)  $\equiv$  Las existencias vendidas totales de un item no pueden exceder el stock del item.

$(\forall p: \text{persona})(p \in \text{claves}(e.\text{ventas}) \Rightarrow_L$

$((\forall i: \text{item})((\exists n: \text{nat})(0 \leq n < \text{long}(\text{obtener}(p, e.\text{ventas}())) \wedge_L \pi_1(\text{obtener}(p, e.\text{ventas}())[n]) = i) \Rightarrow_L$

$\text{cantVendidaPuesto}(\text{claves}(e.\text{ventas}), i) \leq \text{obtener}(i, e.\text{stock})))$

(7) y (8)  $\equiv$  Cada iterador de la venta de una persona que sea sin descuento, debe apuntar a una venta de esa persona con una cantidad o item asociado para los cuales un descuento no es aplicable; además, el item asociado al iterador debe ser el mismo que esta asociado a la venta de la persona.

$(\forall p: \text{persona})(p \in \text{claves}(e.\text{ventasSinDesc}) \Rightarrow_L$   
 $((\forall i: \text{item})(\text{definido?}(i, \text{obtener}(p, e.\text{ventasSinDesc})) \Rightarrow_L$   
 $((\forall itL: \text{itBi}(\text{secu}(\langle \text{item}, \text{cant} \rangle))) (\text{está?}(itL, \text{obtener}(i, \text{obtener}(p, e.\text{ventasSinDesc})))) \Rightarrow_L$   
 $(\neg \text{definido?}(i, e.\text{descuentos}) \vee_L \text{obtener}(i, e.\text{descuentos})[\pi_2(\text{siguiente}(itL))]=0) \wedge (\pi_1(\text{siguiente}(itL)=i))))))$

(9)  $\equiv$  Cada iterador de la venta de una persona que sea sin descuento, esta asociado unívocamente a una venta de la persona.

$(\forall p: \text{persona})(p \in \text{claves}(e.\text{ventasSinDesc}) \Rightarrow_L$   
 $((\forall i: \text{item})(\text{definido?}(i, \text{obtener}(p, e.\text{ventasSinDesc})) \Rightarrow_L$   
 $((\forall itL1, itL2: \text{itBi}(\text{secu}(\langle \text{item}, \text{cant} \rangle)))$   
 $(\text{está?}(itL1, \text{obtener}(i, \text{obtener}(p, e.\text{ventasSinDesc}))) \wedge (\text{está?}(itL2, \text{obtener}(i, \text{obtener}(p, e.\text{ventasSinDesc}))) \wedge$   
 $itL1 \neq itL2) \Rightarrow_L$   
 $((\exists i, j: \text{nat})((0 \leq i, j < \text{long}(\text{obtener}(p, e.\text{ventas}))) \wedge i \neq j) \wedge_L$   
 $\text{siguiente}(itL1)=\text{obtener}(p, e.\text{ventas})[i] \wedge \text{siguiente}(itL2)=\text{obtener}(p, e.\text{ventas})[j])))$

$\text{gastoPuestoPersona} : \text{dicc}(\text{item} \times \text{secu}(\langle \text{cant} \times \text{bool} \rangle)) \times \text{dicc}(\text{item} \times \text{precio}) \times \longrightarrow \text{dinero}$   
 $\text{dicc}(\text{item} \times \text{ad}(\text{nat}))$   
 $\text{gastoPuestoPersona}(vp, m, d) \equiv \text{if } \text{vacía?}(vp) \text{ then}$   
 $0$   
 $\text{else}$   
 $\text{aplicarDescuento}(\text{obtener}(\pi_1(\text{prim}(vp))), e.\text{descuentos}[\pi_2(\text{prim}(vp))]) \times \pi_2(vp) +$   
 $\text{gastoPuestoPersona}(\text{fin}(vp), m, d)$   
 $\text{fi}$   
 $\text{cantVendidaPuesto} : \text{conj}(\text{persona}) \times \text{item} \longrightarrow \text{cant}$   
 $\text{cantVendidaPuesto}(cp, i) \equiv \text{if } \emptyset?(cp) \text{ then}$   
 $0$   
 $\text{else}$   
 $\text{cantVendidaPersona}(\text{obtener}(\text{dameUno}(cp), e.\text{ventas}), i) +$   
 $\text{cantVendidaPuesto}(\text{sinUno}(cp), i)$   
 $\text{fi}$   
 $\text{cantVendidaPersona} : \text{secu}(\langle \text{item} \times \text{cant} \rangle) \times \text{item} \longrightarrow \text{cant}$   
 $\text{cantVendidaPersona}(vp, i) \equiv \text{if } \text{vacía?}(vp) \text{ then}$   
 $0$   
 $\text{else}$   
 $\text{if } \pi_1(\text{prim}(vp)) = i \text{ then}$   
 $\pi_2(\text{prim}(vp)) + \text{cantVendidaPersona}(\text{fin}(vp), i)$   
 $\text{else}$   
 $\text{cantVendidaPersona}(\text{fin}(vp), i)$   
 $\text{fi}$   
 $\text{fi}$   
 $\text{Abs} : \text{estr } e \longrightarrow \text{puesto} \quad \{ \text{Rep}(e) \}$

$\text{Abs}(e) \equiv p: \text{puesto} /$   
 $\text{menu}(p) =_{\text{obs}} \text{claves}(e.\text{menu}) \wedge$   
 $\text{stock}(p) =_{\text{obs}} \text{claves}(e.\text{stock}) \wedge_L$   
 $(\forall i: \text{item})(i \in \text{menu}(p) \wedge \text{precio}(p, i) =_{\text{obs}} \text{obtener}(e.\text{menu}, i) \wedge \text{stock}(p, i) =_{\text{obs}} \text{obtener}(e.\text{stock}, i)) \wedge$   
 $(\forall i: \text{item})(\text{def?}(i, e.\text{descuentos}) \Rightarrow_L ((\forall c: \text{cant})(0 \leq c < \text{long}(\text{obtener}(i, e.\text{descuentos}))) \Rightarrow_L \text{descuento}(p, i, c)$   
 $=_{\text{obs}} \text{obtener}(i, e.\text{descuentos}[c]))) \wedge$   
 $(\forall i: \text{item})(\neg \text{def?}(i, e.\text{descuentos}) \Rightarrow_L ((\forall c: \text{cant})(\text{descuento}(p, i, c) =_{\text{obs}} 0))) \wedge$   
 $(\forall a: \text{persona})(a \in \text{claves}(e.\text{ventas}) \Rightarrow_L \text{ventas}(p, a) =_{\text{obs}} \text{listaAMulticonj}(\text{obtener}(e.\text{ventas}, a)))$

## 2.4. Algoritmos

### Algoritmos

---

```

iCrearPuesto(in  $m$ : dicc(item,nat), in  $s$ : dicc(item,nat), in  $d$ : dicc(item,dicc(cant,nat)))  $\rightarrow res$ : puesto
1: itD  $\leftarrow$  CrearIt(d)  $\triangleright \mathcal{O}(1)$ 
2: descuentos  $\leftarrow$  Vacio()  $\triangleright \mathcal{O}(1)$ 
3: while HaySiguiente(itD) do  $\triangleright \mathcal{O}(I)$ 
4:   stockInicial  $\leftarrow$  Significado(s, SiguienteClave(itD))  $\triangleright \mathcal{O}(\log(I))$ 
5:   arr  $\leftarrow$  arreglo_dimensionable[1...stockInicial]  $\triangleright \mathcal{O}(\text{stockInicial})$ 
6:   Definir(descuentos, SiguienteClave(itD), arr)  $\triangleright \mathcal{O}(\log(I))$ 
7:   c  $\leftarrow$  Significado(d, SiguienteClave(itD))  $\triangleright \mathcal{O}(\log(I))$ 
8:   itC  $\leftarrow$  CrearIt(c)  $\triangleright \mathcal{O}(1)$ 
9:   i  $\leftarrow$  1  $\triangleright \mathcal{O}(1)$ 
10:  minDescuento  $\leftarrow$  SiguienteSignificado(itC)  $\triangleright \mathcal{O}(1)$ 
11:  while i  $\leq$  stockInicial do  $\triangleright \mathcal{O}(\text{stockInicial})$ 
12:    if i < minDescuento then  $\triangleright \mathcal{O}(1)$ 
13:      arr[i]  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
14:    else
15:      if Definido?(c, i) then  $\triangleright \mathcal{O}(\log(\text{cant}))$ 
16:        arr[i]  $\leftarrow$  Significado(c, i)  $\triangleright \mathcal{O}(\log(\text{cant}))$ 
17:        ultI  $\leftarrow$  i  $\triangleright \mathcal{O}(1)$ 
18:      else
19:        arr[i]  $\leftarrow$  Significado(c, ultI)  $\triangleright \mathcal{O}(\log(\text{cant}))$ 
20:      end if
21:    end if
22:    i  $\leftarrow$  i + 1  $\triangleright \mathcal{O}(1)$ 
23:  end while
24:  Avanzar(itD)  $\triangleright \mathcal{O}(1)$ 
25: end while
26: res  $\leftarrow$  ( m, s, Vacio(), descuentos, Vacio(), Vacio() )  $\triangleright \mathcal{O}(1)$ 
  Complejidad:  $\mathcal{O}(I + \text{stockInicial} + \log(\text{cant}))$ 
  Justificación: Los iteradores de los diccionarios recorren inorden.

```

---



---

```

iVender(in  $p$ : puesto, in  $a$ : persona, in  $i$ : item, in  $c$ : cant)
1: if Definido?(p.gastosDe, a) then
2:   nuevoGasto  $\leftarrow$  Significado(p.gastosDe, a) + Precio(p, i)  $\times$  (1 - (Descuento(p, i, c) / 100))  $\triangleright$ 
    $\mathcal{O}(\log(A) + \log(I))$ 
3: else
4:   nuevoGasto = Precio(p, i)  $\times$  (1 - (Descuento(p, i, c) / 100))  $\triangleright \mathcal{O}(\log(I))$ 
5: end if
6: Definir(p.gastosDe, a, nuevoGasto)  $\triangleright \mathcal{O}(\log(A))$ 
7: ventas  $\leftarrow$  Significado(p.ventas, a)  $\triangleright \mathcal{O}(\log(A))$ 
8: it  $\leftarrow$  AgregarAtras(ventas, <i, c>)  $\triangleright \mathcal{O}(1)$ 
9: if Descuento(p, i, c) = 0 then  $\triangleright \mathcal{O}(\log(I))$ 
10:  ventasSinDescuento  $\leftarrow$  Significado(Significado(p.ventasSinDesc, a), i)  $\triangleright \mathcal{O}(\log(A) + \log(I))$ 
11:  AgregarAtras(ventasSinDescuento, it)  $\triangleright \mathcal{O}(1)$ 
12: end if
  Complejidad:  $\mathcal{O}(\log(A) + \log(I))$ 

```

---

---

**iMenu**(in  $p$ : puesto)  $\rightarrow res$ : conj(item)

|  |                                 |
|--|---------------------------------|
| 1: $it \leftarrow \text{CrearIt}(p.\text{menu})$ | $\triangleright \mathcal{O}(1)$ |
| 2: $\text{conjItems} \leftarrow \text{Vacio}()$  | $\triangleright \mathcal{O}(1)$ |
| 3: <b>while</b> HaySiguiente(it) <b>do</b>       | $\triangleright \mathcal{O}(I)$ |
| 4:     AgregarRapido(conjItems, Siguiente(it))   | $\triangleright \mathcal{O}(1)$ |
| 5:     Avanzar(it)                               | $\triangleright \mathcal{O}(1)$ |
| 6: <b>end while</b>                              |                                 |
| 7: $res \leftarrow \text{conjItems}$             | $\triangleright \mathcal{O}(1)$ |
| <u>Complejidad:</u> $\mathcal{O}(I)$             |                                 |

---



---

**iStock**(in  $p$ : puesto, in  $i$ : item)  $\rightarrow res$ : cant

|   |                                       |
|---|---------------------------------------|
| 1: $res \leftarrow \text{Significado}(p.\text{stock}, i)$ | $\triangleright \mathcal{O}(\log(I))$ |
| <u>Complejidad:</u> $\mathcal{O}(\log(I))$                |                                       |

---



---

**iPrecio**(in  $p$ : puesto, in  $i$ : item, in  $i$ : item)  $\rightarrow res$ : nat

|  |                                       |
|--|---------------------------------------|
| 1: $res \leftarrow \text{Significado}(p.\text{menu}, i)$ | $\triangleright \mathcal{O}(\log(I))$ |
| <u>Complejidad:</u> $\mathcal{O}(\log(I))$               |                                       |

---



---

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

|   |                                       |
|---|---------------------------------------|
| 1: <b>if</b> Definido?(p.descuentos, i) <b>then</b>               | $\triangleright \mathcal{O}(\log(I))$ |
| 2: $res \leftarrow \text{Significado}(p.\text{descuentos}, i)[c]$ | $\triangleright \mathcal{O}(\log(I))$ |
| 3: <b>else</b>  |                                       |
| 4: $res \leftarrow 0$   | $\triangleright \mathcal{O}(1)$       |
| 5: <b>end if</b>  |                                       |
| <u>Complejidad:</u> $\mathcal{O}(\log(I))$                        |                                       |

---



---

**iGastosDe**(in  $p$ : puesto, in  $a$ : persona)  $\rightarrow res$ : nat

|  |                                       |
|--|---------------------------------------|
| 1: $res \leftarrow \text{Significado}(p.\text{gastosDe}, a)$ | $\triangleright \mathcal{O}(\log(A))$ |
| <u>Complejidad:</u> $\mathcal{O}(\log(A))$                   |                                       |

---

### 3. Módulo Cola de Prioridad Acotada( $\langle\alpha, \beta\rangle$ )

El módulo Cola de Prioridad Acotada ( $\langle\alpha, \beta\rangle$ ) proporciona una estructura de datos que permite almacenar elementos y acceder a ellos según su prioridad. Esta implementación utiliza un vector para representar el heap, una estructura de árbol binario completa.

En esta implementación, los elementos se almacenan en el vector heap de acuerdo con su prioridad, de manera que el elemento de mayor prioridad se encuentra en la posición raíz del árbol binario. La propiedad del heap asegura que para cada nodo, el valor de su padre es mayor o igual que los valores de sus hijos.

#### 3.1. TAD Cola de Prioridad Acotada( $\gamma$ )

**TAD COLA DE PRIORIDAD ACOTADA( $\gamma$ )**

**igualdad observacional**

$$(\forall c, c' : \text{colaPriorA}(\gamma)) \left( c =_{\text{obs}} c' \iff \left( \begin{array}{l} \text{cota}(c) =_{\text{obs}} \text{cota}(c') \wedge \text{vacía?}(c) =_{\text{obs}} \text{vacía?}(c') \wedge_{\text{L}} \\ (\neg \text{vacía?}(c) \Rightarrow_{\text{L}} (\text{próximo}(c) =_{\text{obs}} \text{próximo}(c') \wedge \\ \text{desencolar}(c) =_{\text{obs}} \text{desencolar}(c'))) \end{array} \right) \right)$$

**parámetros formales**

**géneros**  $\gamma$

**operaciones**  $\bullet < \bullet : \gamma \times \gamma \longrightarrow \text{bool}$

Relación de orden total estricto<sup>1</sup>

**géneros**  $\text{colaPriorA}(\gamma)$

**exporta**  $\text{colaPriorA}(\gamma)$ , generadores, observadores

**usa** **BOOL**

**observadores básicos**

$\text{cota} : \text{colaPriorA}(\gamma) \longrightarrow \text{nat}$

$\text{vacía?} : \text{colaPriorA}(\gamma) \longrightarrow \text{bool}$

$\text{próximo} : \text{colaPriorA}(\gamma) \ c \longrightarrow \gamma$

$\{\neg \text{vacía?}(c)\}$

$\text{desencolar} : \text{colaPriorA}(\gamma) \ c \longrightarrow \text{colaPriorA}(\gamma)$

$\{\neg \text{vacía?}(c)\}$

**generadores**

$\text{vacía} : \text{nat } \text{cota} \longrightarrow \text{colaPriorA}(\gamma)$

$\{\text{cota} > 0\}$

$\text{encolar} : \gamma \ e \times \text{colaPriorA}(\gamma) \ c \longrightarrow \text{colaPriorA}(\gamma)$

$\{\text{respetarCota}(c, e)\}$

**otras operaciones**

$\text{longitud} : \text{colaPriorA}(\gamma) \longrightarrow \text{nat}$

**axiomas**  $\forall c: \text{colaPriorA}(\gamma), \forall e: \gamma, \forall n: \text{nat}$

$\text{cota}(\text{vacía}(n)) \equiv n$

$\text{cota}(\text{encolar}(e, c)) \equiv \text{cota}(c)$

$\text{vacía?}(\text{vacía}(n)) \equiv \text{true}$

$\text{vacía?}(\text{encolar}(e, c)) \equiv \text{false}$

$\text{próximo}(\text{encolar}(e, c)) \equiv \text{if } \text{vacía?}(c) \vee_{\text{L}} \text{próximo}(c) < e \text{ then } e \text{ else } \text{próximo}(c) \text{ fi}$

$\text{desencolar}(\text{encolar}(e, c)) \equiv \text{if } \text{vacía?}(c) \vee_{\text{L}} \text{próximo}(c) < e \text{ then } c \text{ else } \text{encolar}(e, \text{desencolar}(c)) \text{ fi}$

$\text{longitud}(c) \equiv \text{if } \text{vacía?}(c) \text{ then } 0 \text{ else } 1 + \text{longitud}(\text{desencolar}(c)) \text{ fi}$

**Fin TAD**

<sup>1</sup>Una relación es un orden total estricto cuando se cumple:

**Antirreflexividad:**  $\neg a < a$  para todo  $a : \gamma$

**Antisimetría:**  $(a < b \Rightarrow \neg b < a)$  para todo  $a, b : \gamma, a \neq b$

**Transitividad:**  $((a < b \wedge b < c) \Rightarrow a < c)$  para todo  $a, b, c : \gamma$

**Totalidad:**  $(a < b \vee b < a)$  para todo  $a, b : \gamma$

## 3.2. Interfaz

### Interfaz

parámetros formales

géneros  $\alpha$

función  $\bullet < \bullet$ (in  $e_1 : \langle \alpha, \beta \rangle$ , in  $e_2 : \langle \alpha, \beta \rangle$ )  $\rightarrow res$  función COPIAR(in  $e : \langle \alpha, \beta \rangle$ )  $\rightarrow res : \langle \alpha, \beta \rangle$   
 $: \text{bool}$  **Pre**  $\equiv \{\text{true}\}$  **Post**  $\equiv \{res =_{\text{obs}} e\}$   
**Pre**  $\equiv \{\text{true}\}$  **Complejidad:**  $\mathcal{O}(\text{copy}(e))$   
**Post**  $\equiv \{res =_{\text{obs}} (e_1 < e_2)\}$  **Descripción:** función de copia de  $\langle \alpha, \beta \rangle$ 's  
**Complejidad:**  $\mathcal{O}(e_1 < e_2)$   
**Descripción:** relación de orden de  $\langle \alpha, \beta \rangle$ 's

se explica con: COLA DE PRIORIDAD ACOTADA( $\alpha$ )

géneros: colaPriorA( $\langle \alpha, \beta \rangle$ ).

Operaciones básicas de cola de prioridad Acotada

VACÍA()  $\rightarrow res : \text{colaPriorA}(\langle \alpha, \beta \rangle)$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{vacía}\}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** genera una cola vacía.

ENCOLAR(in/out  $c : \text{colaPriorA}(\langle \alpha, \beta \rangle)$ , in  $e : (\langle \alpha, \beta \rangle)$ )  $\rightarrow res : \text{puntero}(\alpha)$

**Pre**  $\equiv \{c =_{\text{obs}} c_0\}$

**Post**  $\equiv \{c =_{\text{obs}} \text{encolar}(e, c_0)\}$

**Complejidad:**  $\mathcal{O}(\log(\#c))$

**Descripción:** Encola un elemento en la cola de prioridad. El elemento se agrega al final del vector que representa el heap y luego se restaura la propiedad del heap.

DESENCOLAR(in/out  $c : \text{colaPriorA}(\langle \alpha, \beta \rangle)$ )

**Pre**  $\equiv \{c =_{\text{obs}} c_0 \wedge \neg \text{vacía?}(c)\}$

**Post**  $\equiv \{c =_{\text{obs}} \text{desencolar}(e, c_0)\}$

**Complejidad:**  $\mathcal{O}(\log(\#c))$

**Descripción:** Elimina el elemento de mayor prioridad de la cola. El elemento de la raíz se intercambia con el último elemento en el vector, se elimina del vector y luego se restaura la propiedad del heap.

PRÓXIMO(in  $c : \text{colaPriorA}(\langle \alpha, \beta \rangle)$ )  $\rightarrow res : \alpha$

**Pre**  $\equiv \{c =_{\text{obs}} c_0\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{próximo?}(c)\}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** devuelve el elemento de mayor prioridad en la cola.

VACÍA?(in  $c : \text{colaPriorA}(\langle \alpha, \beta \rangle)$ )  $\rightarrow res : \text{bool}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{vacía?}(c)\}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** devuelve un booleano diciendo si la cola es o no vacía.



### 3.3. Representación

## Representación

Representación de la cola de prioridad acotada

La cola de prioridad acotada se representa con un vector, que representa a un heap. La cota representa la cantidad de elementos maxima que puede tener la cola de prioridad.

$\text{colaPriorA}(\langle \alpha, \beta \rangle)$  se representa con *estr*

donde *estr* es  $\text{tupla}(\text{heap: arreglo\_dimensionable de } \langle \mathbf{a}: \alpha, \mathbf{b}: \text{itDiccLog} \rangle, \text{indices: diccLog}(\beta, \text{nat}) , \text{longitud: nat} , \text{cota: nat} )$

$\text{Rep} : \text{colaPriorA}(\alpha) \longrightarrow \text{bool}$

$\text{Rep}(e) \equiv \text{true} \iff$

$(\forall i \in [0, \text{Longitud}(c) - 1], \text{ If } 2i + 1 < \text{Longitud}(c), \text{ then } c[i] \geq c[2i + 1] \wedge$   
 $\forall i \in [0, \text{Longitud}(c) - 1], \text{ If } 2i + 2 < \text{Longitud}(c), \text{ then } c[i] \geq c[2i + 2]) \wedge$   
 $(c.\text{longitud} \leq c.\text{cota}) \wedge$   
 $(\forall b : \beta)(\text{Definido?}(c.\text{indices}, b) \Rightarrow_{\text{L}} (\exists i : \text{nat})(0 \leq i < c.\text{longitud} \wedge_{\text{L}} *c.\text{heap}[i].b =$   
 $\text{Significado}(c.\text{indices}, b)) \wedge$   
 $(\forall t : \langle \alpha, \beta \rangle)(\exists i : \text{nat})(0 \leq i < c.\text{longitud} \wedge_{\text{L}} c.\text{heap}[i] = \text{tupla}) \Rightarrow_{\text{L}} (\exists b : \beta)(\text{Definido?}(c.\text{indices}, b) \wedge_{\text{L}}$   
 $\text{Significado}(c.\text{indices}, b) = \text{tupla}.b))$

$\text{Abs} : \text{estr } e \longrightarrow \text{colaPriorA}(\langle \alpha, \beta \rangle)$

$\{\text{Rep}(e)\}$

$\text{Abs}(e) \equiv c: \text{colaPriorA}(\langle \alpha, \beta \rangle) /$

$\text{vacía?}(c) = (\text{e.longitud} = 0) \wedge (\neg \text{vacía?}(c) \Rightarrow \text{próximo}(c) = \langle c.\text{heap}[0].\mathbf{a}, \pi_1(\text{Actual}(c.\text{heap}[0].\mathbf{b})) \rangle \wedge$   
 $\text{desencolar}(c) = \text{arrayACola}(\text{e.heap}, \text{e.indices}, \text{e.longitud}))$

$\text{arrayACola} : \text{ad}(\langle \alpha \times \text{itDiccLog} \rangle) \times \text{diccLog}(\beta \times \text{nat}) \times \text{nat} \longrightarrow \text{colaPriorA}(\langle \alpha, \beta \rangle)$

$\text{arrayACola}(a, d, l) \equiv \text{if } l = 0 \text{ then}$

$\text{vacía}$

**else**

$\text{encolar}(\langle \pi_1(a[l - 1]), \pi_2(\text{Actual}(\pi_2(a[l - 1]))) \rangle, \text{arrayACola}(a, d, l - 1))$

**fi**

### 3.4. Algoritmos

## Algoritmos

---



---

|  |                                 |
|--|---------------------------------|
| <b>•&gt;•</b> (in $e1: \langle a: \alpha, b: \beta \rangle$ ), in $e2: \langle a: \alpha, b: \beta \rangle \rightarrow res: bool$<br>1: $res \leftarrow (e1.a > e2.a \vee (e1.a = e2.a \wedge e1.b > e2.b))$<br><u>Complejidad:</u> $\mathcal{O}(1)$ | $\triangleright \mathcal{O}(1)$ |
|--|---------------------------------|

---



---

|   |                                 |
|---|---------------------------------|
| <b>iVacía</b> (in $cota: nat \rightarrow res: colaPriorA(\langle \alpha, \beta \rangle)$<br>1: $res \leftarrow \langle arreglo\_dimensionable(0..cota - 1), Vacio(), 0, cota \rangle$<br><u>Complejidad:</u> $\mathcal{O}(1)$ | $\triangleright \mathcal{O}(1)$ |
|---|---------------------------------|

---



---

|   |   |
|---|---|
| <b>iEncolar</b> (in/out $c: colaPriorA(\langle \alpha, \beta \rangle)$ , in $e: \langle alfa: \alpha, beta: \beta \rangle \rightarrow res: puntero(\alpha)$<br>1: <b>if</b> Definido?(c.indices, e.beta) <b>then</b><br>2: $i \leftarrow Significado(c.indices, e.beta)$<br>3: <b>if</b> $e > c.heap[i]$ <b>then</b><br>4: $index \leftarrow HeapifyUp(c, i)$<br>5: <b>else</b><br>6: $index \leftarrow HeapifyDown(c, i)$<br>7: <b>end if</b><br>8: <b>else</b><br>9: $c.longitud \leftarrow c.longitud + 1$<br>10: $it \leftarrow Definir(c.indices, e.beta, c.longitud - 1)$<br>11: $c.heap[c.longitud - 1] \leftarrow \langle e.alfa, it \rangle$<br>12: $index \leftarrow HeapifyUp(c, c.longitud - 1)$<br>13: <b>end if</b><br>14: $puntero \leftarrow \&c.heap[index]$<br>15: $res \leftarrow puntero$<br><u>Complejidad:</u> $\mathcal{O}(\log(\#c) + copy(e))$<br><u>Justificación:</u> En el peor caso, la cantidad de claves en el diccionario c.indices es igual a $\#c$ , donde $\#c$ es c.longitud. | $\triangleright \mathcal{O}(1)$<br>$\triangleright \mathcal{O}(1)$<br>$\triangleright \mathcal{O}(\log(\#c))$<br>$\triangleright \mathcal{O}(\log(\#c))$<br>$\triangleright \mathcal{O}(1)$<br>$\triangleright \mathcal{O}(1)$<br>$\triangleright \mathcal{O}(1)$<br>$\triangleright \mathcal{O}(1)$<br>$\triangleright \mathcal{O}(1)$ |
|---|---|

---



---

|   |   |
|---|---|
| <b>iDesencolar</b> (in/out $c: colaPriorA(\langle \alpha, \beta \rangle)$<br>1: $c.heap[0] \leftarrow c.heap[c.longitud - 1]$<br>2: $c.longitud \leftarrow c.longitud - 1$<br>3: $c.heap \leftarrow HeapifyDown(c.heap, 0)$<br><u>Complejidad:</u> $\mathcal{O}(\log(\#c))$ | $\triangleright \mathcal{O}(1)$<br>$\triangleright \mathcal{O}(1)$<br>$\triangleright \mathcal{O}(\log(\#c))$ |
|---|---|

---



---

|  |                                 |
|--|---------------------------------|
| <b>iPróximo</b> (in $c: colaPriorA(\langle \alpha, \beta \rangle) \rightarrow res: \alpha$<br>1: $res \leftarrow \langle c.heap[0].a, SiguienteClave(c.heap[0].b) \rangle$<br><u>Complejidad:</u> $\mathcal{O}(1)$ | $\triangleright \mathcal{O}(1)$ |
|--|---------------------------------|

---



---

|   |                                 |
|---|---------------------------------|
| <b>iVacía?</b> (in $c: colaPriorA(\langle \alpha, \beta \rangle) \rightarrow res: bool$<br>1: $res \leftarrow (c.longitud = 0)$<br><u>Complejidad:</u> $\mathcal{O}(1)$ | $\triangleright \mathcal{O}(1)$ |
|---|---------------------------------|

---

### 3.5. Funciones auxiliares

#### Funcion privada: HeapifyUp

**Descripción:** Esta función se utiliza después de encolar un elemento en el heap y se encarga de mantener la propiedad del heap llamada 'orden del padre'. Compara el elemento en la posición `index` con su padre y, si el elemento es mayor, los intercambia. Luego, se repite este proceso de comparación e intercambio ascendiendo en el árbol hasta que el elemento se encuentra en la posición correcta.

**Pre**  $\equiv$  {El índice debe estar dentro de los límites del vector heap.}

**Post**  $\equiv$  {El elemento en la posición índice se ha colocado en la posición correcta de acuerdo con la propiedad del heap 'orden del padre'. Devuelve el índice de la posición correcta.}

---

---

**iHeapifyUp**(in/out  $c$ : colaPriorA( $\langle \alpha, \beta \rangle$ ), in  $i$ : nat)  $\rightarrow res$ : nat

```

1: if i > 0 then
2:   indexPadre  $\leftarrow \lfloor (i - 1) / 2 \rfloor$   $\triangleright \mathcal{O}(1)$ 
3:   while c.heap[i] > c.heap[indexPadre] do  $\triangleright \mathcal{O}(\log(\#c))$ 
4:     indexPadre  $\leftarrow \lfloor (i - 1) / 2 \rfloor$   $\triangleright \mathcal{O}(1)$ 
5:     temp  $\leftarrow$  c.heap[i]  $\triangleright \mathcal{O}(1)$ 
6:     c.heap[i]  $\leftarrow$  c.heap[indexPadre]  $\triangleright \mathcal{O}(1)$ 
7:     c.heap[indexPadre]  $\leftarrow$  temp  $\triangleright \mathcal{O}(1)$ 
8:     i  $\leftarrow$  indexPadre  $\triangleright \mathcal{O}(1)$ 
9:     SiguienteSignificado(c.heap[i].b) = i  $\triangleright \mathcal{O}(1)$ 
10:    SiguienteSignificado(c.heap[indexPadre].b) = indexPadre  $\triangleright \mathcal{O}(1)$ 
11:   end while
12: end if
13: res  $\leftarrow$  i  $\triangleright \mathcal{O}(1)$ 

```

Complejidad:  $\mathcal{O}(\log(\#c))$

Justificación: Donde  $\#c$  es la logitud de c.heap.

---

### Funcion privada: HeapifyDown

**Descripción:** Esta función se utiliza después de desencolar un elemento del heap y se encarga de mantener la propiedad del heap llamada 'orden del hijo'. Compara el elemento en la posición índice con sus hijos y, si alguno de los hijos es mayor, intercambia el elemento con el hijo de mayor valor. Luego, se repite este proceso de comparación e intercambio descendiendo en el árbol hasta que el elemento se encuentra en la posición correcta.

**Pre**  $\equiv$  {El índice debe estar dentro de los límites del vector heap.}

**Post**  $\equiv$  {El elemento en la posición índice se ha colocado en la posición correcta de acuerdo con la propiedad del heap 'orden del hijo'. Devuelve el índice de la posición correcta.}

---

**iHeapifyDown**(in/out  $c: \text{colaPriorA}(\langle \alpha, \beta \rangle)$ , in  $i: \text{nat}$ )  $\rightarrow res: \text{nat}$

---

```

1: largo  $\leftarrow$  c.longitud  $\triangleright \mathcal{O}(1)$ 
2: while i < largo do  $\triangleright \mathcal{O}(\log(\#c))$ 
3:   hijoIzq  $\leftarrow$  2  $\times$  i + 1  $\triangleright \mathcal{O}(1)$ 
4:   hijoDer  $\leftarrow$  2  $\times$  i + 2  $\triangleright \mathcal{O}(1)$ 
5:   máximo  $\leftarrow$  i  $\triangleright \mathcal{O}(1)$ 
6:   if hijoIzq < largo  $\wedge$  c.heap[hijoIzq] > c.heap[máximo] then  $\triangleright \mathcal{O}(1)$ 
7:     SiguienteSignificado(c.heap[maximo]) = hijoIzq  $\triangleright \mathcal{O}(1)$ 
8:     SiguienteSignificado(c.heap[hijoIzq]) = maximo  $\triangleright \mathcal{O}(1)$ 
9:     máximo  $\leftarrow$  hijoIzq  $\triangleright \mathcal{O}(1)$ 
10:  else
11:    if hijoDer < largo  $\wedge$  c.heap[hijoDer] > c.heap[máximo] then  $\triangleright \mathcal{O}(1)$ 
12:      SiguienteSignificado(c.heap[maximo]) = hijoDer  $\triangleright \mathcal{O}(1)$ 
13:      SiguienteSignificado(c.heap[hijoDer]) = maximo  $\triangleright \mathcal{O}(1)$ 
14:      máximo  $\leftarrow$  hijoDer  $\triangleright \mathcal{O}(1)$ 
15:    end if
16:  end if
17:  if máximo != i then  $\triangleright \mathcal{O}(1)$ 
18:    temp  $\leftarrow$  c.heap[i]  $\triangleright \mathcal{O}(1)$ 
19:    c.heap[i]  $\leftarrow$  c.heap[máximo]  $\triangleright \mathcal{O}(1)$ 
20:    c.heap[máximo]  $\leftarrow$  temp  $\triangleright \mathcal{O}(1)$ 
21:  else
22:    break
23:  end if
24: end while
25: res  $\leftarrow$  máximo

```

Complejidad:  $\mathcal{O}(\log(\#c))$

Justificación: Donde  $\#c$  es la logitud de c.heap.

---