

Algoritmos y Estructuras de Datos 2

Final 2021-03-04

1. Ejercicio 1

1.a.

- **Los productos comprados son válidos:** Todos los productos de todas las compras son claves definidas del diccionario `productos1` (los productos comprados se identifican por código). La vuelta no es necesaria porque es válido tener productos definidos que jamás fueron vendidos.
- **Las cantidades compradas son válidas:** Todas las cantidades de todas las compras deben ser mayor o igual a 1.
- **La cantidad total de ventas por producto se corresponde con las ventas registradas:** La cantidad total de ventas de un producto en particular es la suma de la cantidad vendida de ese producto entre todas las compras de todas las personas. A su vez debe valer la vuelta, todos los productos vendidos deben estar definidos en el diccionario `ventas_por_prod`.
- **El mapeo entre el código del producto y su nombre es biyectivo:** El diccionario `productos1` debe ser el inverso de `productos2` (las claves pasan a ser significados). Necesitamos que todos los códigos mapeen a su nombre y que todos los nombres mapeen a su código.

```
Rep: estr → bool
Rep(e) ≡ true ⇔ (
  ComprasVálidasYContabilizadas(e)
  ∧ VentasPorProductoVálidas(e)
  ∧ e.productos1 =obs InvertirDicc(e.productos2)
)

ComprasVálidasYContabilizadas: estr → bool
ComprasVálidasYContabilizadas(e) ≡ true ⇔ (
  (∀p: persona)(p ∈ claves(e.compras) ⇒ (
    (∀c: compra)(c ∈ obtener(p, e.compras) ⇒ (
      c.cant ≥ 1
      ∧ def?(c.cod_prod, e.productos1)
      ∧ def?(c.cod_prod, e.ventas_por_prod)
    ))
  ))
)

VentasPorProductoVálidas: estr → bool
VentasPorProductoVálidas(e) ≡ true ⇔ (
  (∀cp: cod_prod)(cp ∈ claves(e.ventas_por_prod) ⇒ (
    obtener(cp, e.ventas_por_prod) = ContarVentas(cp, e.compras)
  ))
)

ContarVentas: cod_prod × dicc(persona → conj(compra)) → nat
Sea p = dameUno(claves(d))
Sea cs = obtener(p, d)
ContarVentas(cp, d) ≡
  if vacío?(claves(d)) then
    0
  else
    ContarVentasAux(cp, cs) + ContarVentas(cp, borrar(p, d))
  fi

ContarVentasAux: cod_prod × conj(compra) → nat
```

```

Sea c = dameUno(cs)
ContarVentasAux(cp, cs) ≡
    if vacío?(cs) then
        0
    else
        if c.cod_prod = cp then c.cant else 0 fi + ContarVentasAux(cp, sinUno(cs))
    fi

InvertirDicc: dicc(K → V) → dicc(V → K)
Sea k = dameUno(claves(d))
Sea v = obtener(k, d)
InvertirDicc(d) ≡
    if vacío?(claves(d)) then
        vacío
    else
        definir(v, k, InvertirDicc(borrar(k, d)))
    fi

```

1.b.

El diccionario `productos1` mapea el código de producto a su nombre. Los códigos seguramente tengan una cota ya que no hay infinitos productos en un supermercado. Por lo tanto se podría representar el diccionario con un HashTable para que las operaciones sean $O(1)$. Si la cota resulta demasiado grande y los códigos no son densos (hay muchos códigos sin usar), pero se determina que un porcentaje alto de ellos sí son densos y tienen una cota más chica, se podría representar el diccionario con 2 estructuras: un HashTable para los códigos densos el cuál se usa únicamente si el código es inferior a la cota, y caso contrario el resto de los códigos se colocan en un árbol AVL donde las operaciones resultan $O(\log(m))$ donde m es la cantidad de productos que tienen un código que supera la cota.

El diccionario `productos2` mapea el nombre del producto a su código. En este caso podríamos representar el diccionario con un Trie ya que esta estructura es eficiente para operar con claves de tipo string. En el peor caso, las operaciones tendrían una complejidad de $O(\log(k))$ donde k es el nombre de producto más largo.

2. Ejercicio 2

operacion es tupla $\langle \text{diaCompra: nat, diaVenta: nat} \rangle$

OptimizarOperación(in s: arreglo(nat)) → out res: operacion

1: res ← OptimizarOperaciónAux(s, 1, tam(s))

Complejidad: $O(n \log(n))$

OptimizarOperaciónAux(in s: arreglo(nat), in low: nat, in high: nat) → out res: operacion

```

1: if high - low = 0 then
2:   res ← ⟨ diaCompra: low, diaVenta: low ⟩
3: else
4:   mid ← (low + high) / 2
5:   opIzq ← OptimizarOperaciónAux(s, low, mid)
6:   opDer ← OptimizarOperaciónAux(s, mid + 1, high)
7:   opMid ← ⟨ diaCompra: BuscarMinPrecio(s, low, mid), diaVenta: BuscarMaxPrecio(s, mid+1, high) ⟩
8:   if Ganancia(s, opIzq) > Ganancia(s, opMid) ∧ Ganancia(s, opIzq) > Ganancia(s, opDer) then
9:     res ← opIzq
10:  else if Ganancia(s, opMid) > Ganancia(s, opDer) then
11:    res ← opMid
12:  else
13:    res ← opDer
14:  end if
15: end if

```

Complejidad: $O(n \log(n))$

Ganancia(**in** s: arreglo(nat), **in** o: operacion) \rightarrow **out** res: nat

1: res \leftarrow s[o.diaVenta] - s[o.diaCompra]

Complejidad: $O(1)$

3. Ejercicio 3

4. Ejercicio 4

5. Ejercicio 5