

# Soluciones de los Ejercicios, MARP I

Mario Calvarro Marines



# Índice general

---

<b>1. Hoja 1</b>	<b>5</b>
1.1. Ejercicio 1. (Iker Muñoz)	5
1.2. Ejercicio 2. (Sergio García)	5
1.3. Ejercicio 3. (Alba Bautista)	5
1.4. Ejercicio 4. (Juan Fonseca)	6
1.5. Ejercicio 5. (Marta Vicente)	6
1.6. Ejercicio 6. (Juan Diego Barrado)	6
1.7. Ejercicio 7. (Leonardo Macías)	7
1.8. Ejercicio 8. (Beñat Pérez)	7
1.9. Ejercicio 9 (Lucía Alonso)	9
1.10. Ejercicio 10 (Javier Saras)	10
1.11. Ejercicio 11 (Javier Amado)	10



# Hoja 1

---

## Ejercicio 1. (Iker Muñoz)

No será coste constante amortizado. Contraejemplo:

```
fun multiapilar (p: pila, k: nat)
  mientras k > 0 hacer
    apilar (p)
    k = k - 1
  fmientras
ffun
```

Coste  $O(k)$ . Si consideramos la secuencia de  $n$  llamadas a multiapilar tendremos:

$$\hat{c}_i = \frac{1}{n} \sum_{i=1}^n c_i = \frac{1}{n} \sum_{i=1}^n k = k.$$

## Ejercicio 2. (Sergio García)

Sí:

$$\underbrace{1 \dots \underbrace{10 \dots 0}_{k-1}}_n \xrightarrow{\text{incrementar}} \underbrace{1 \dots \underbrace{01 \dots 1}_{k-1}}_n \xrightarrow{\text{decrementar}} \underbrace{1 \dots \underbrace{10 \dots 0}_{k-1}}_n \rightarrow \dots n \text{ operaciones.}$$

Incrementar y decrementar:  $O(k) \Rightarrow n$  operaciones  $O(nk)$ .

## Ejercicio 3. (Alba Bautista)

Hemos de calcular  $\frac{\sum_{j=1}^n c_j}{j}$ . Lo hacemos primero para  $j = 2^n$ .

Consideramos  $C_{in} = \{2^i : 0 \leq i \leq n\}$  y  $\overline{C}_{in} = (1, \dots, 2^n) \setminus C_n$ .

$$\begin{aligned} \sum_{i=1}^{\infty} C_i &= \sum_{i \in C_n} C_i + \sum_{i=1}^n C_i = \sum_{i=1}^{\infty} 2^i + \sum_{i \in C_n} 1 = 2^{n+1} - 1 + |\overline{C}_n| = |1 \dots 2^n| - |C_n| = \\ &= 2^n - (n+1) \Rightarrow \sum_{i=1}^j C_i = (2^{n+1} - 1) + (2^n - n - 1) = 3 \cdot 2^n - n - 2 \Rightarrow \frac{\sum_{i=1}^j C_i}{2^n} \leq 3 \in O(1) \end{aligned}$$

De forma similar si tomamos  $j = 2^n + j' : j' \in 1, \dots, 2^n - 1$ .

$$\Rightarrow \sum_{i=0}^j 3 \cdot 2^n - n - 2 + j \Rightarrow \frac{\sum_{i=0}^j C_i}{j} \leq 3 \in O(1)$$

Ya que  $\begin{cases} 2^n \leq j \\ j' \leq j \end{cases}$

## Ejercicio 4. (Juan Fonseca)

Utilizamos el método del potencial para calcular el coste amortizado:

$\Phi(D_i)$  elementos en la lista tras operación  $i$ -ésima

- Añadir un número:

$$\hat{c}_i = c_1 + \Phi(D_i) - \Phi(D_{i-1}) = 1 + (k+1) - k = 2 \in O(1)$$

- Reducir-lista:

$$\hat{c}_i = c_1 + \Phi(D_i) - \Phi(D_{i-1}) = k+1+1-k = 2 \in O(1)$$

## Ejercicio 5. (Marta Vicente)

```

proc contar (C[0, ..., k-1] de {0, 1}, E/S posSig: nat)
  j := 0
  mientras j < k AND j < posSig AND C[j] = 1
    C[j] := 0
    j := j + 1
  fmientras

  si j < k AND j < posSig
    C[j] := 1
  fsi

  si j < k AND j = posSig
    C[j] := 1
    posSig := posSig + 1
  fsi
fproc

proc resetear (E/S posSig: nat)
  posSig := 0
fproc

```

$$valor(c) = \sum_{j=0}^{\text{posSig}} 2^j C[j]$$

## Ejercicio 6. (Juan Diego Barrado)

DISCLAMER: Este ejercicio no está ni de cerca completo. Recomendando no utilizar este documento para estudiarlo.

Buscar:

Para cada  $A_i$ , hacer búsqueda binaria tiene coste en  $O(\log 2^i) = O(i)$ .

En el caso peor, el elemento que buscamos está en  $A_{k-1}$ , luego  $\sum_{i=0}^{k-1} i = \frac{(k-1)k}{2} \in O(k^2) = O(\log^2(n))$ .

Insertar: Cuando insertamos  $n$  elementos como mucho viajan al nivel  $\log(n)$  vaciamos  $A_0$  de 1 elemento  $\frac{n}{2}$  veces,  $A_1$ , 2 elementos  $\frac{n}{4}$  veces y  $A_j$ ,  $2^j$  elementos  $\frac{n}{2^{j+1}}$  veces  $\Rightarrow$

$$T(n) = \sum_{j=0}^{\log n} 2^j \cdot \frac{n}{2^{j+1}} = \frac{n}{2} \log n.$$

## Ejercicio 7. (Leonardo Macías)

Base:  $b[0, \dots, n-1] \in \mathbb{Z}^n$  con  $b[i] \geq 2$ .

Contador:  $v[0, \dots, n-1]$  con  $0 \leq v[i] < b[i]$  y  $v[0]$  menos significativo.

```
proc incrementar (b: base, v: contador)
  i := 0
  mientras i < n AND v[i] = b[i] - 1 hacer
    v[i] := 0
    i := i + 1
  fmientras
  si i < n hacer
    v[i] := v[i] + 1
  fsi
fproc
```

$v[0]$  cambia  $n$  veces,  $v[1]$  cambia  $\frac{n}{b[0]}$  veces,  $\dots$ ,  $v[i]$  cambia  $\frac{n}{\prod_{j=0}^i b[j]}$  veces.

M. Agregación:  $N^0$  Cambios:  $\sum_{i=0}^n \frac{n}{\prod_{j=0}^i b[j]}$

Como  $\forall j \in \{0, \dots, n-1\} : b[j] \geq 2$ :

$$\sum_{i=0}^n \frac{1}{\prod_{j=0}^i b[j]} \leq \sum_{i=0}^n \frac{1}{2^i} \leq \sum_{i=0}^{\infty} \frac{1}{2^i} = 2$$

Coste amortizado:

$$\frac{1}{n} \sum_{i=0}^n \frac{n}{\prod_{j=0}^i b[j]} = \sum_{i=0}^n \frac{1}{\prod_{j=0}^i b[j]} \leq 2$$

## Ejercicio 8. (Beñat Pérez)

a) Utilizamos dos pilas una de los elementos y otra de máximos.

Operaciones:

```
proc apilar(p1, p2: pila; k: ent)
  si !p2.empty AND k <= p2.top =>
    p2.push(k)
  fsi
  p1.push(k)
fproc
```

```

proc maximo (p2: pila; k: ent)
  si !p2.empty =>
    return p2.top
  fsi
fproc

```

```

proc desapilar(p1, p2: pila)
  si !p1.empty =>
    si p1.top = p2.top =>
      p2.pop
    fsi
  fsi
  p1.pop
fproc

```

b) Operaciones:

```

proc anyadir(p1: pila; k: ent)
  si !p1.empty =>
    p1.push(k, max(k, p1.top))
  si no =>
    p1.push(k, max)
  fsi
fproc

```

```

func max(p1, p2: pila)
  return max(p1.top.b, p2.top.b)
fproc

```

```

proc desapilar(p1, p2: pilas; k: ent)
  si !p2.empty =>
    si !p1.empty =>
      p2.push(p1.top.a, p1.top.a)
      p1.pop
      mientras !p1.empty hacer
        p2.push(p1.top.a, max(p1.top.a, p2.top.b))
        p1.pop
      fmientras
      p2.pop
    fsi
  si no =>
    p2.pop
  fsi
fproc

```

En el caso peor tendremos  $O(n)$ . Sin embargo, amortizado de  $n$  operaciones nos saldrá  $O(1)$  por operación.

Por el método de contabilidad. Asumamos que apilar tiene coste 3 (poner el elemento en  $p_1$ , quitarlo de  $p_1$  y ponerlo en  $p_2$ ). Así, solamente nos queda que desapilar tiene coste 1, ya que habría que hacer una sola operación.

Además, la función máximo es constante.

Con todo, nos queda que el coste amortizado es  $O(1)$ .



## Ejercicio 9 (Lucía Alonso)

Tenemos una cola  $C$ , creamos una doble cola llamada *maximos* para llevar el control del máximo de  $C_1$ :

$$\text{maximos}[i] : 0 \leq i < n : \forall j : i < j \leq n : \text{maximos}[i] \leq \text{maximos}[j]$$

- En la llamada a *borrar*() elemento de  $C_1$  debemos comprobar si es 1<sup>er</sup> máximo.
- En la llamada a *insertar*() elemento de  $C_1$  debemos borrar todos los elementos de máximo menores que él.

Ejemplo:

$C = 5, 4, 3, 2, 1; \text{ maximos} = 5, 4, 3, 2, 1 \Rightarrow \text{insertar}(3)$   
 $C = 5, 4, 3, 2, 1, 3; \text{ maximos} = 5, 4, 3, 3 \Rightarrow \text{remove}()$   
 $C = 4, 3, 2, 1, 3; \text{ maximos} = 4, 3, 3 \Rightarrow \text{insert}(6)$   
 $C = 4, 3, 2, 1, 3, 6; \text{ maximos} = 6.$

```
proc insertar (cola: C1, doble cola : maximos, elemento: x)
  c.push(x)
  mientras !maximos.empty() AND maximos.back() < x
    maximo.pop_back()
  fmientras
  maximos.push_back(x)
fproc

proc borrar (cola: C1, doble cola: maximos)
  si c.front() = maximos.front()
    maximos.pop_front()
  fsi
  c.pop()
fproc

{!maximo.empty()}
func getMax(doble cola: maximo)
  return maximo.front()
ffunc
```

Coste amortizado. Por el método del potencial.

- Caso peor: Se añade el máximo de  $C$  y *maximos* está ordenada decrecientemente:

$$c_i = 4 + n \in O(n) \Rightarrow \text{Lineal.}$$

En la siguiente llamada, cola vacía:

$$c_{i+1} = 4 + 1 \in O(1) \Rightarrow \text{cte.}$$

- Método del potencial:

$$\Phi(D_i) = \text{longitud de la doble cola de máximos.}$$

Al empezar con  $C$  vacía se cumple  $\forall i : \Phi(D_i) - \Phi(D_0) > 0$ .

- En la llamada a *getMax*():

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 \in O(1)$$

Al ser una operación de consulta.

- En la llamada *borrar*():

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = \begin{cases} 2 & \text{si no se borra el máximo} \\ 2 + m - 2 - m = 1 & \text{c.c} \end{cases}$$

- En la llamada a *insertar*():

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 4 + b_i + (k - b_i) - k = 4 \in O(1)$$

Es decir, el coste amortizado es constante.

## Ejercicio 10 (Javier Saras)

Por el método del potencial.

$$\Phi(D_i) = M + m$$

Con  $\Phi(D_0) = 0$  y  $\Phi(D_i) \geq 0$ .

- *maximo*():

$$\hat{c}_i = 1 + M + m - M - m = 1$$

- *minimo*():

$$\hat{c}_i = 1 + M + m - M - m = 1$$

- *eliminar*():

- Máximo:  $\hat{c}_i = 2 + M - 1 + m - M - m = 1$ .
- Mínimo:  $\hat{c}_i = 2 + M + m - 1 - M - m = 1$ .
- Ninguno:  $\hat{c}_i = 1 + M + m - M - m = 1$ .

- *insertar*():

- Máximo:  $\hat{c}_i = m + 3 + 1 + m + 1 - (M + m) = 5$
- Mínimo:  $\hat{c}_i = m + 3 + M + 1 + 1 - (M + m) = 5$
- Ninguno:  $\hat{c}_i = a + b + 3 + M - a + 1 + m - b + 1 - (M + m) = 5$

## Ejercicio 11 (Javier Amado)

```
class multiconj {
    valores: array <int>
    n: int
    +constructor vacio() {
        valores := new int[max]
        n := 0;
    }

    +insertar(v: int) {
        si (n < max) {
            valores[n] = v
            n := n + 1
        }
        fsi
    }
    +elimMayores {
        nborrar := ceil(n/2)
```

```

m := mediana(valores/n)
i := 0; elim = 0; j = 0;
mientras i < n
    si valores[i] > m
        valores[i] := null
        elim := elim + 1
    fsi
    i := i + 1
fmientras
mientras j < n AND elim < nborrar
    si valores[j] = m
        valores[j] := null
        elim := elim + 1
    fsi
    j := j + 1
fmientras
i := 0; j := 0
mientras j < n
    si valores[j] != null
        valores[i] = valores[j]
        i := i + 1
    fsi
fmientras
n := n - elim; f elim mayores?
}
}

```

Coste por el método del potencial

$$\Phi(D_i) = 2n$$

$$\hat{c}_{insertar} = c_{insertar} + (\Phi(D_i) - \Phi(D_{i-1})) = 1 + 2 = 3 \in O(1)$$

$$\hat{c}_{elimMayores} = c_{elimMayores} + (\Phi(D_i) - \Phi(D_{i-1})) = n - 2 \lfloor n/2 \rfloor = \begin{cases} 0 & \text{si } n \text{ es par} \\ 1 & \text{si } n \text{ es impar} \end{cases}$$