

Soluciones de los Ejercicios, MARP I

Mario Calvarro Marines

Índice general

1. Hoja 1	5
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. (??)	6
1.5. Ejercicio 5. (??)	6
1.6. Ejercicio 6. (Juan Diego Barrado)	6
1.7. Ejercicio 7. (Leonardo Macías)	7
1.8. Ejercicio 8. (Beñat Pérez)	7

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. (??)

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. (??)

```

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 $O(1)$.