# Estructuras de Datos y Algoritmos

## Grados en Ingeniería Informática, de Computadores y del Software

Examen Parcial, 11 de Febrero de 2013.

1. **Diseño iterativo (4 puntos)** Especificar, diseñar, verificar y calcular el coste de un algoritmo que, dado un vector $v$ de enteros que puede ser vacío ($n \geq 0$) y que tan solo puede contener los valores 0 y 1, devuelva un booleano que indique si el vector tiene la forma:

$$\boxed{1\,|\,1\,|\cdots|\,1\,|\,1\,|\,0\,|\,0\,|\cdots|\,0\,|\,0}$$

Las dos zonas de ceros y unos pueden tener cualquier longitud, incluida la nula.

*Sugerencia (no penaliza si no se sigue):* la verificación puede resultar más fácil si se elige un invariante de la forma:

$$\boxed{1\,|\cdots|\,1\,|\;?\,?\,?\,?\,?\;|\,0\,|\cdots|\,0}$$

2. **Diseño recursivo (4 puntos)** Se define el *histograma* de un vector $v$ de enteros como otro vector $w$ que contiene en la posición $i$ la suma $v[0]+\ldots+v[i]$. Así, por ejemplo, el vector $\{1, 2, 4, 3, -2\}$ tendría como histograma $\{1, 3, 7, 10, 8\}$.

Se pide especificar, diseñar, demostrar la corrección y calcular el coste de un algoritmo recursivo, lo más eficiente posible, que dado un vector de enteros que puede ser vacío devuelva su histograma.

3. **Diseño TADs (2 puntos)** Diseñar un TAD *Polinomio* que permita manejar polinomios con coeficientes enteros en una indeterminada. Ejemplos de polinomios serían $3x^4+1$, $-2x^{99}+5x^2$, ó $x^{507}+x^{200}+x$. El tipo *Polinomio* deberá permitir, además de la construcción de polinomios, su suma, resta y multiplicación (que devolverán nuevos polinomios); su evaluación para una $x$ concreta; y la posibilidad de verificar si un polinomio está vacío o es igual a otro dado.

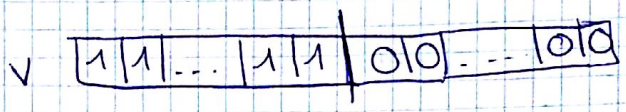
Se pide:

- Elegir un *tipo representante* para el TAD eficiente en tiempo y espacio, suponiendo que los polinomios están formados por $k$ o menos monomios (elementos de la forma $c \cdot x^n$), donde tanto $c$ como $n$ pueden llegar a ser muy grandes (aunque quepan en un `int` estándar). La constante $k$ debe formar parte de la implementación de tu TAD.
- Dar el *invariante de la representación* y la *relación de equivalencia* de la representación elegida.
- Implementar el archivo `polinomio.h` con la definición de la clase. Se debe incluir las operaciones públicas necesarias para manejar el tipo, ya sean constructoras, observadoras o modificadoras; y la parte privada con la representación. Para cada operación debe darse su precondición y postcondición.

**Nota:** No hace falta que implementes ninguna de estas operaciones; basta con que las declares y describas correctamente.

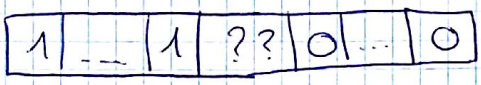
# Feb 2013

① Algoritmo que detecte




---

Invariante




Especificación:

$\{ n \geq 0 \wedge n = long(v) \wedge \forall w : 0 \leq w < n \; (v[w] = 0 \vee v[w] = 1) \}$

fun unosCeros (int v[], int n) return bool b;

$\{ b = [\exists i : 0 \leq i < n : [(\forall t : 0 \leq t < i : v[t] = 1) \wedge (\forall s : i \leq s < n : v[s] = 0)] \wedge$

$\wedge [(\# w : 0 \leq w < n \cdot v[w] = 1) = (\# r : 0 \leq r < n \cdot v[r] = 0)]$

Diseño:

```
bool unosCeros (int v[], int n){
    bool b;
    int i;
    b = true;
    i = 0;
    while (i < n-i){
        b = b && (v[i] == 1) && (v[n-i-1] == 0)
        i = i+1;
    }
    return b;
}
```

Ejemplos

v | 1 | 1 | 0 | 1 | n=4

| i | n-i-1 | v[i] | v[n-i-1] | b |
|---|---|---|---|---|
| 0 | 3 | 1 | 1 | t |
| 1 | 2 | 1 | 0 | f |
| 2 | | | | (f) |

- Si n es impar

v | 1 | 1 | 1 | 0 | 0 | n=5

| i | n-i-1 | v[i] | v[n-i-1] | b |
|---|---|---|---|---|
| 0 | 4 | 1 | 0 | t |
| 1 | 3 | 1 | 0 | t |
| 2 | 2 | 1 | 1 | t (f) |

## Verificación

### Invariante

$I \equiv (0 \leq 2i \leq n+1) \wedge (\forall w : 0 \leq w < i : (v[w] = 1 \wedge v[n-w-1] = 0))$

$B \equiv 2i < n$

**I1** $\{P\}\ S_0\ \{I\}$

$\{P\}\ b = t\ ;\ i = 0\ \{I\}$

$pwd(b = t\ ;\ i = 0\ ;\ I) \Leftrightarrow I\Big|_{i}^{0}\Big|_{b}^{true} \Leftrightarrow$

$\Leftrightarrow (0 \leq 0 \leq n) \wedge (\underbrace{\forall w : 0 \leq w < 0 : \ldots}_{false} \overbrace{= true}^{true}) \Leftrightarrow (0 \leq n) \Leftarrow P$

**I2** $\{I \wedge B\}\ b = b\ \&\&\ v[i] == 1\ \&\&\ v[n-1-i] == 0\ ;\ i = i+1\ \{I\}$

$pwd(\_) \Leftrightarrow I\Big|_{i}^{i+1}\Big|_{b}^{b \wedge \ldots}$

$\Leftrightarrow (0 \leq 2(i+1) \leq n+1 \wedge ((\forall w : 0 \leq w < i+1 : v[w] = 1 \wedge v[n-w-1] = 0) \wedge v[i] = 1 \wedge v[n-i-1] = 0)$
$= b \wedge v[i] == 1 \wedge v[n-i-1] = 0)$

$\Leftrightarrow (0 \leq 2(i+1) \leq n+1 \wedge (b = (\forall w : 0 \leq w < i : v[w] = 1 \wedge v[n-w-1] = 0)$

$I \wedge B \equiv (0 \leq 2 \leq n+1) \wedge (b = (\forall w : 0 \leq w < i : v[w] = 1 \wedge v[n-w-1] = 0) \wedge (2i < n)$

$0 \leq 2i \Rightarrow 0 \leq 2(i+1)$

$2i < n \Rightarrow 2i+1 \leq n \Rightarrow 2i+2 \leq n+1$

$2i < n \Rightarrow 2i+2 \leq n+2 \Rightarrow 2i+2 \leq n+1$ //

**I3** $I \wedge \neg B \Rightarrow Q$

$(0 \leq 2i \leq n+1) \wedge (b = \ldots) \wedge (2i \geq n)$

$\Rightarrow (n \leq 2i \leq n+1) \wedge (b = \ldots)$

$\Rightarrow [(2i = n) \vee (2i = n+1)] \wedge (b = \ldots)$

$\Rightarrow [(2i = n) \wedge (b = \ldots)] \vee [(2i = n+1) \wedge (b = \ldots)]$

$\Rightarrow [(2i = n) \wedge (b = \forall w : 0 \leq w < \frac{n}{2} : v[w] = 1 \wedge v[n-w-1] = 0)]$

$\vee [(2i = n+1) \wedge (b = \forall w : 0 \leq w < \frac{n+1}{2} : v[w] = 1 \wedge v[n-w-1] = 0)]$

$\Rightarrow (2i = n) \wedge (b = \forall w : 0 \leq w < \frac{n}{2} : v[w] = 1 \wedge v[n-w-1] = 0)$

$\Leftrightarrow Q.$

Función de cota

$C(v, n, b, i) = n - 2i$

**C1** $I \wedge B \Rightarrow C \geq 0$

$(0 \leq 2i \leq n+1) \wedge (\ldots) \wedge (2i < n)$

$\Rightarrow 0 < n - 2i$

$\Rightarrow 0 \leq n - 2i$

**C2** $\{I \wedge B \wedge C = T\}\ S\ \{C < T\}$

$pmd(\ldots) \Leftrightarrow (n - 2i < T)\Big|_{i}^{i+1}\Big|_{b}^{b \wedge \ldots}$

$\Leftrightarrow (n - 2(i+1) < T)$

$(0 \leq 2i \leq n+1) \wedge (b = \ldots) \wedge (2i < n) \wedge (n - 2i = T)$

$\Rightarrow n - 2i - 2 < T$

$\Rightarrow n - 2(i+1) < T$ ✓

# Complejidad

```
bool ...
    b = true;
    i = 0;
    while (i < n-i)
        b = b && v[i] == 1 && v[n-i-1] == 0;
        i = i + 1;
    }
    return b;
}
```

$T(n) = 11\,\frac{n}{2}$ ; si n par

$T(n) = 11\,\frac{n+1}{2}$ ; si n impar $\Big\} \in \Theta(n)$

n par: $n/2$ vueltas

| i | n-i |
|---|---|
| 0 | 2 |
| ~~1~~ | ~~1~~ |

$n = 2$

n impar: $(n+1)/2$ vueltas

$n = 3$

| i | n-i |
|---|---|
| 0 | 3 |
| 1 | 2 |

## ② Histograma

Dado un vector $v[N]$, su histograma es otro vector $h[n]$ tq $h[i] = v[0] + \_ + v[i]$

### Especificación

$P \equiv \{0 \leq N \wedge long(v) = N\}$

func histograma (int v[], int N) return (int h[])

$Q = \{\forall i : 0 \leq i < N : h[i] = \sum w : 0 \leq w \leq i \; v[w]\}$

### Generalizamos

$P \equiv \{0 \leq N \wedge long(v) = N \wedge long(h) = N \wedge 0 \leq k < N \wedge$
$\wedge sumParc = \sum_{i=0}^{k-1} v[i]\}$

proc histograma (int v[], int N, int k, int h[], int sumP)

$Q = \{\forall i : 0 \leq i < N : h[i] = \sum w : 0 \leq w \leq i \; v[w]\}$

### Diseño

```
void histograma (int v[], int N, int k, int h[], int SumP){
// if (k == N) Nada (nos hemos salido del vector)
if (k<n){
SumaP = SumaP + v[k];
h[k] = SumaP;
histograma (v, N, k+1, h, SumP)
}
```

falta * int histograma

### Complejidad

$T(0) = 0,$

$T(n) = 6 + T(n-1);$

$T(n) = a\,T(n-b) + c\,n^k$

- a: nº de llamadas recursivas
- b: decremento
- $c\,n^k$: complej. de lo que no es llamada recursiva

• Paso 1: Cálculo para un caso partic. $n \in \mathbb{Z}$

$T(n) = 6 + T(n-1)$

$= 6 + 6 \cdot T(n-1) = 2 \cdot 6 + T(n-2)$

$= 2 \cdot 6 + 6 + T(n-3) = 3 \cdot 6 + T(n-3)$

$= \dots$ n pasos $\dots$

$= 6n + T(0) \to 0$

$= 6n$

• Paso 2: Conjetura

$T(n) = 6n$

- CASO BASE $(n=0)$ $T(0) = 0 = 6 \cdot 0$ ✓

lo supongo para $n' < n$

- CASO GENERAL

Hip $\forall n' < n \quad T(n') = 6n'$

$T(n) = 6 + T(n-1)$

$= 6 + 6(n-1) = 6n$ ✓