Taller 1 – Recursión

Curso: Fundamentos de Programación Funcional y Concurrente
Tema: Procesos recursivos e iterativos
Estudiantes:

- Jorge Luis Junior Lasprilla Prada - 2420662
- Andrés Gerardo González Rosero- 2416541

---

1. Informe de procesos

1.1 `maxLin`

- **Tipo de proceso:** recursivo lineal.
- **Descripción:** la función compara el primer elemento de la lista con el máximo de la cola, reduciendo el problema en cada paso.
- **Comportamiento:** se genera un proceso que consume espacio en la pila de llamadas proporcional al tamaño de la lista (`O(n)`).

1.2 `maxIt`

- **Tipo de proceso:** iterativo (implementado mediante recursión de cola).
- **Descripción:** mantiene un acumulador (`currentMax`) que conserva el máximo de los elementos vistos hasta el momento.
- **Comportamiento:** gracias a la recursión de cola, Scala optimiza la ejecución en forma de ciclo, evitando desbordes de pila. Complejidad temporal `O(n)` y espacial `O(1)`.

1.3 `movsTorresHanoi`

- **Tipo de proceso:** recursivo lineal.
- **Descripción:** cada llamada para `n` discos invoca una sola llamada recursiva para `n-1` discos y realiza operaciones aritméticas.
- **Comportamiento:** genera un proceso lineal, análogo a la definición recursiva de la recurrencia `M(n) = 2M(n-1) + 1`.

1.4 `torresHanoi`

- **Tipo de proceso:** recursivo de árbol.

- **Descripción:** cada llamada con `n` discos se descompone en dos llamadas con `n-1`, más un movimiento directo.

- **Comportamiento:** genera un árbol binario de llamadas, de tamaño `2^n - 1`. Esto refleja directamente la estructura del problema.

---

2. Demostraciones de corrección

2.1 `maxLin` (recursiva lineal)

Se desea calcular el número máximo de una lista de números enteros positivos, no vacía.

Sea $L : \text{List}[\mathbb{N}] \to [\mathbb{N}]$ y sea $\mathcal{F}_c$ la función principal del programa, la cual debe satisfacer:

$$\mathcal{F}_c(L) = \max(L)$$

donde $\max(L)$ denota el elemento máximo de L.

Definiciones auxiliares

- **intMax**:
$$\text{intMax}(x, y) = \begin{cases} x & \text{si } x \geq y \\ y & \text{en otro caso} \end{cases}$$

- **maxLin**:
$$\text{maxLin}(L) = \begin{cases} \text{head}(L) & \text{si tail}(L) = \emptyset \\ \text{intMax}(\text{head}(L), \text{maxLin}(\text{tail}(L))) & \text{en otro caso} \end{cases}$$

Lema 1: Correctitud de intMax Para todo $x, y \in \mathbb{N}$:

$$\text{intMax}(x, y) = \max(x, y)$$

Demostración: Por casos triviales de la definición.

Demostración principal por inducción estructural

Base de inducción (Caso base) Sea $L = [a]$ (lista con un elemento). Entonces $\text{tail}(L) = \emptyset$, luego:

$$\text{maxLin}(L) = \text{head}(L) = a$$

Por otro lado, $\max(L) = a$. Por lo tanto:

$$\text{maxLin}(L) = \max(L)$$

Hipótesis inductiva (HI) Supongamos que para toda lista L' de longitud $k \geq 1$:

$$\text{maxLin}(L') = \max(L')$$

Paso inductivo Sea $L = a :: L'$, donde L' es no vacía y de longitud k. Por definición:

$$\text{maxLin}(L) = \text{intMax}(a, \text{maxLin}(L'))$$

Por HI, como L' tiene longitud k:

$$\text{maxLin}(L') = \max(L')$$

Luego:

$$\text{maxLin}(L) = \text{intMax}(a, \max(L'))$$

Por Lema 1:

$$\text{intMax}(a, \max(L')) = \max(a, \max(L'))$$

Por definición de máximo de una lista:

$$\max(a :: L') = \max(a, \max(L'))$$

Por lo tanto:

$$\text{maxLin}(L) = \max(a :: L') = \max(L)$$

Conclusión Por el principio de inducción estructural, para toda lista no vacía L:

$$\text{maxLin}(L) = \max(L)$$

2.2 `maxIt` (iterativa)

Sea L una lista no vacía de números enteros positivos. La función debe satisfacer:

$$\text{maxIt}(L) = \max(L)$$

Donde $\max(L)$ denota el elemento máximo de L.

Demostración por inducción estructural

Definiciones formales:

1. Función principal

$$\text{maxIt}(L) = \text{iter}(\text{tail}(L), \text{head}(L))$$

2. Función auxiliar

$$\text{iter}(R, M) = \begin{cases} M & \text{Si } R = \text{Nil} \\ \text{iter}(\text{tail}(R), \max(\text{head}(R), M)) & \text{Si } R \neq \text{Nil} \end{cases}$$

3. Función máxima

$$\max(L) = \begin{cases} a_1 & \text{si } L = [a_1] \\ \max(a_1, \max(a_2 :: ... :: a_n)) & \text{Si } L = a_1, a_2, :: ... ::, a_n \end{cases}$$

Lema 1: Propiedad fundamental de iter Para cualquier lista R y cualquier entero M:

$$\text{iter}(R, M) = \max(M, \max(R))$$

Demostración del lema 1 por inducción estructural sobre R.

Caso base: $R = \text{Nil}$

$$\text{iter}(\text{Nil}, M) = M = \max(M, -\infty) = \max(M, \max(\text{Nil}))$$

Caso inductivo: $R = r :: rs$

Hipótesis inductiva: $\text{iter}(rs, M') = \max(M', \max(rs))$ para cualquier M'

$$\text{iter}(r :: rs, M) = \text{iter}(rs, \max(r, M))$$

Por hipótesis inductiva:

$$= \max(\max(r, M), \max(rs))$$

Por propiedades del máximo:

$$\max(\max(r, M), \max(rs)) = \max(M, \max(r :: rs))$$

Teorema principal: Correctitud de maxIt Para toda lista no vacía L:

$$\text{maxIt}(L) = \max(L)$$

Demostración Sea $L = a_1 :: a_2 :: ... :: a_n :: \text{Nil}$ con $n \geq 1$

Por definición:

$$\text{maxIt}(L) = \text{iter}(\text{tail}(L), \text{head}(L)) = \text{iter}(a_2 :: ... :: a_n :: \text{Nil}, a_1)$$

Por el Lema 1:

$$= \max(a_1, \max(a_2 :: ... :: a_n :: \text{Nil}))$$

Por definición de max:

$$= \max(a_1 :: a_2 :: ... :: a_n :: \text{Nil}) = \max(L)$$

Lema 2: Terminación La función maxIt termina para todas las listas no vacías.

Demostración Definamos $\text{long}(R)$ como la longitud de la lista R.

Para la función auxiliar $\text{iter}(R, M)$

- Caso base $R = \text{Nil}$ termina inmediatamente
- Paso Recursivo: $\text{iter}(r :: rs, M)$ llama a $\text{iter}(rs, \max(r, M))$

Se cumple:

$$\text{long}(rs) = \text{long}(r :: rs) - 1 < \text{long}(r :: rs)$$

Como $\text{long}(R) \in \mathbb{N}$ y disminuye estrictamente en cada llamada recursiva, eventualmente se alcanzará el caso base $R = \text{Nil}$.

Para $\text{maxIt}(L)$, como L es no vacía:

$$\text{long}(\text{tail}(L)) = \text{long}(L) - 1 \geq 0$$

$\therefore$ La función termina.

2.3 `movsTorresHanoi` (recursiva lineal)

Sea f : $\rightarrow$ la función que devuelve el número mínimo de movimientos para resolver Hanoi.

Queremos demostrar:
n 1 : movsTorresHanoi(n) == f(n)

Caso base:
$n = 1$
$movsTorresHanoi(1) \rightarrow 1$

Por definición, $f(1) = 1$
Luego: $movsTorresHanoi(1) == f(1)$

Paso inductivo:
Supongamos que $movsTorresHanoi(k) = f(k) = 2^k - 1$.

Para n = k+1:
$movsTorresHanoi(k+1) \rightarrow 2 * movsTorresHanoi(k) + 1$
$\rightarrow 2 * (2^k - 1) + 1 = 2^{(k+1)} - 1$
$= f(k+1)$

Conclusión: por inducción matemática, movsTorresHanoi es correcta.

2.4 `torresHanoi` (recursiva de árbol)

Sea f : $\rightarrow$ List[(Int,Int)] la función que devuelve la secuencia de movimientos correcta para Hanoi.

Queremos demostrar:
$\forall n \geq 1 : torresHanoi(n, t1, t2, t3) == f(n, t1, t2, t3)$

Caso base:
$n = 1$
$torresHanoi(1, t1, t2, t3) \rightarrow List((t1, t3))$
Por definición, ese es el único movimiento correcto.

Paso inductivo:
Supongamos que $torresHanoi(k, ...)$ devuelve correctamente la secuencia para k discos.

Para $n = k + 1$:
$torresHanoi(k+1, t1, t2, t3) \rightarrow torresHanoi(k, t1, t3, t2) + +List((t1, t3)) + +torresHanoi(k, t2, t1, t3)$

- La primera lista mueve k discos de origen a auxiliar (correcto por HI).
- El elemento central mueve el disco mayor de origen a destino (correcto por definición).
- La última lista mueve k discos de auxiliar a destino (correcto por HI).

La concatenación respeta el orden de Hanoi y produce una solución válida.

Conclusión: por inducción matemática, torresHanoi es correcta.

---

3. Casos de prueba

3.1 `maxLin`

Entrada	Salida esperada	Salida obtenida	Comentario
`[5]`	5	5	Caso base: un solo elemento
`[3,2,9,1]`	9	9	Máximo en el medio
`[10,10,10]`	10	10	Todos los elementos iguales
`[1,2,3,4,5]`	5	5	Lista creciente
`[100,2,99,3]`	100	100	Máximo al inicio

3.2 `maxIt`

Entrada	Salida esperada	Salida obtenida	Comentario
`[5]`	5	5	Caso base
`[3,2,9,1]`	9	9	Coincide con maxLin
`[10,10,10]`	10	10	Elementos repetidos
`[1,2,3,4,5]`	5	5	Lista creciente
`[100,2,99,3]`	100	100	Coincide con maxLin

3.3 `movsTorresHanoi`

Entrada	Salida esperada	Salida obtenida	Comentario
1	1	1	Caso base
2	3	3	
3	7	7	
4	15	15	
5	31	31	
64	(número grande)	18446744073709551615	Comparar con siglos (ejemplo guía)

El numero obtenido al correrlo con 64 discos es 18446744073709551615, lo que en siglos es 5849424173

3.4 `torresHanoi`

Entrada	Salida esperada	Salida obtenida	Comentario
`(1,1,2,3)`	`List((1,3))`	List((1,3))	Caso base: n=1
`(2,1,2,3)`	`List((1,2), (1,3), (2,3))`	List((1,2), (1,3), (2,3))	Secuencia de 3 movimientos

Entrada	Salida esperada	Salida obtenida	Comentario
`(3,1,2,3)`	(lista de 7 movimientos correcta)	List((1,3), (1,2), (3,2), (1,3), (2,1), (2,3), (1,3))	Verificar contra el enunciado
`(4,1,2,3)`	(lista de 15 movimientos correcta)	List((1,2), (1,3), (2,3), (1,2), (3,1), (3,2), (1,2), (1,3), (2,3),(2,1), (3,1), (2,3), (1,2), (1,3), (2,3))	Verificar contra el enunciado
`(5,1,2,3)`	(lista de 31 movimientos correcta, opcional)	List((1,3), (1,2), (3,2), (1,3), (2,1), (2,3), (1,3), (1,2), (3,2),(3,1), (2,1), (3,2), (1,3), (1,2), (3,2), (1,3), (2,1), (2,3), (1,3), (2,1), (3,2), (3,1), (2,1), (2,3), (1,3), (1,2), (3,2), (1,3), (2,1), (2,3), (1,3))	Verificar contra el enunciado

---

4. Conclusiones

- Se identificaron correctamente los tipos de procesos: recursión lineal, iteración y recursión de árbol.
- Las demostraciones por inducción y por invariante garantizan la corrección de las funciones.
- Los casos de prueba confirman empíricamente los resultados.
- Se evidencia la diferencia entre eficiencia y claridad: la recursión refleja la estructura matemática del problema, mientras que la iteración resulta más eficiente en espacio.