

# Informe sobre taller de colecciones y expresiones for

**Curso:** Fundamentos de Programación Funcional y Concurrente

**Tema:** Reconocimiento de patrones

## Estudiantes:

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

## 1. Informe de uso de colecciones y expresiones for

Función	¿Usa Expresiones For?	¿Usa Colecciones?	Observaciones
<code>subindices</code>	Sí - 2 expresiones for	Sí - Set, Range, List	Genera combinaciones usando for sobre rangos
<code>subSecuenciaAsoc</code>	Sí - 1 expresión for	Sí - Seq, List	Transforma índices a elementos usando for
<code>subSecuenciasDe</code>	Sí - 1 expresión for	Sí - Set, Seq	Itera sobre conjunto de índices con for
<code>Incremental</code>	Sí - 1 expresión for	Sí - Seq, Range	Verifica orden con for sobre índices
<code>subSecuenciasInc</code>	Sí - 1 expresión for	Sí - Set, Seq	Filtra subsecuencias incrementales con for
<code>subsecuenciaIncrementalMasLarga</code>	Sí - 1 expresión for	Sí - Set, Seq	Encuentra máximo con for en lugar de maxBy
<code>ssimlComenzandoEn</code>	Sí - 2 expresiones for	Sí - Range, Seq, IndexedSeq	Búsqueda recursiva con

Función	¿Usa Expresiones For?	¿Usa Colecciones?	Observaciones
			for para candidatos
subSecIncMasLargaV2	Sí - 4 expresiones for	Sí - Map, Range, Seq, IndexedSeq	Versión optimizada con memorización usando for

El uso de colecciones y expresiones for en Scala ha demostrado ser una técnica de programación extremadamente poderosa y expresiva. Durante la implementación de este taller, pude constatar varias ventajas significativas.

## 2. Informe de corrección

### Correctitud de subíndices

- **Caso base:** (  $m = 0$  )
  - Si (  $n - i = 0$  ), entonces (  $m \leq 0$  ).
  - Por definición: (  $\text{subindices}(i, n) = \{ \langle \rangle \}$  )
  - El único subconjunto posible de índices en (  $[i, n)$  ) es la lista vacía.
- **Paso inductivo:** Supongamos cierta (  $P(k)$  ) para todo (  $k < m$  ). Demostremos (  $P(m)$  ), con (  $m = n - i > 0$  ).
  - El algoritmo recorre todos los valores de bitmask desde 0 hasta (  $2^m - 1$  ).
  - Cada bitmask representa una combinación de índices.
  - Toda lista producida es estrictamente creciente.
  - Se generan todas las posibles subsecuencias.
- **Conclusión inductiva:**
  - (  $\forall i, n. \text{subindices}(i, n) = \{ [a_1, \dots, a_k] \mid i \leq a_1 < \dots < a_k < n \}$  )

## Correctitud de `subSecuenciasInc`

- **Caso base:** (  $n = 0$  )
  - (  $\text{subSecuenciasInc}([]) = \{ [] \}$  )
- **Hipótesis inductiva:** La propiedad se cumple para toda secuencia de longitud (  $n$  ).
- **Paso inductivo:** Para (  $s' = [s_0, s_1, \dots, s_n]$  )
  - (  $\text{subSecuenciasInc}(s') = A \cup B$  ), donde:
    - (  $A$  ): subsecuencias que no contienen (  $s_n$  )
    - (  $B$  ): subsecuencias que sí contienen (  $s_n$  ), con último elemento menor que (  $s_n$  )
- **Conclusión:**
  - (  $\text{subSecuenciasInc}(s) = \{ t \in \text{subSecuenciasDe}(s) \mid t \text{ es estrictamente creciente} \}$  )

## Correctitud de `subsecuenciaIncrementalMasLarga`

- **Caso base:** Si no hay subsecuencias incrementales, devuelve (  $[]$  ).
- **Caso general:**
  - (  $\text{subsecuenciaIncrementalMasLarga}(s) = \arg\max_{t \in \text{subSecuenciasInc}(s)} |t|$  )
  - Devuelve la subsecuencia incremental más larga.

## Correctitud de `subSecIncMasLargaV2`

- **Caso base:** (  $s = [] \rightarrow \text{subSecIncMasLargaV2}([]) = []$  )
- **Hipótesis inductiva:** Para toda secuencia (  $s'$  ) de longitud (  $m < n$  ), la función devuelve la SIML.
- **Paso inductivo:**
  - Se construye un mapa (  $\text{memo}[i] =$  ) subsecuencia incremental más larga comenzando en (  $i$  ).
  - Se selecciona la de mayor longitud.
  - (  $\text{subSecIncMasLargaV2}(s) = \text{SIML}(s)$  )

### 3. Casos de prueba

#### Subíndices

Caso	Entrada	Descripción	Resultado Esperado
1	(2, 3)	Índices entre 2 y 2	Set(List(), List(2))
2	(0, 3)	Índices entre 0 y 2	Set(List(), List(0), List(1), List(2), List(0,1), List(0,2), List(1,2), List(0,1,2))
3	(0, 1)	Índices entre 0 y 0	Set(List(), List(0))
4	(1, 4)	Índices entre 1 y 3	Set(List(), List(1), List(2), List(3), List(1,2), List(1,3), List(2,3), List(1,2,3))
5	(0, 0)	Intervalo vacío	Set(List())

#### subSecuenciaAsoc

Caso	Entrada	Descripción	Resultado Esperado
1	(s, Seq())	Secuencia vacía de índices	List()
2	(s, Seq(0, 2, 4))	Índices 0, 2, 4 de s	List(20, 10, 15)
3	(s, Seq(1, 2, 4, 6))	Índices 1, 2, 4, 6 de s	List(30, 10, 15, 17)
4	(s, Seq(0))	Solo el primer elemento	List(20)
5	(s, Seq(3, 5))	Índices 3 y 5 de s	List(40, 16)

#### subSecuenciaDe

Caso	Entrada	Descripción	Resultado Esperado
1	Seq()	Secuencia vacía	Set(List())
2	Seq(10)	Secuencia con un elemento	Set(List(), List(10))
3	Seq(20, 30, 10)	Secuencia de 3 elementos	Set(List(), List(20), List(30), List(10), List(20,30), List(20,10), List(30,10), List(20,30,10))
4	Seq(10, 20)	Secuencia creciente de 2 elementos	Set(List(), List(10), List(20), List(10,20))

Caso	Entrada	Descripción	Resultado Esperado
5	Seq(1, 2, 3)	Secuencia creciente de 3 elementos	Set(List(), List(1), List(2), List(3), List(1,2), List(1,3), List(2,3), List(1,2,3))

## Incremental

Caso	Entrada	Descripción	Resultado Esperado
1	Seq(10, 20, 30)	Secuencia estrictamente creciente	true
2	Seq(10, 20, 15)	Secuencia no creciente	false
3	Seq()	Secuencia vacía	true
4	Seq(5)	Secuencia con un elemento	true
5	Seq(10, 10, 20)	Secuencia con elementos iguales	false

## subSecuenciasInc

Caso	Entrada	Descripción	Resultado Esperado
1	Seq()	Secuencia vacía	Set(List())
2	Seq(10)	Secuencia con un elemento	Set(List(), List(10))
3	Seq(20, 30, 10)	Secuencia con elementos mezclados	Set(List(), List(20), List(30), List(10), List(20,30))
4	Seq(20, 30, 10, 40, 15, 16, 17)	Secuencia compleja	Set(List(), List(20), List(30), List(10), List(40), List(15), List(16), List(17), List(20,30), List(20,40), List(10,40), List(10,15), List(10,16), List(10,17), List(15,16), List(15,17), List(16,17), List(10,15,16), List(10,15,17), List(10,16,17), List(15,16,17), List(10,15,16,17))
5	Seq(50, 40, 30)	Secuencia decreciente	Set(List(), List(50), List(40), List(30))

## subSecuenciaIncrementalMasLarga

Caso	Entrada	Descripción	Resultado Esperado
1	Seq()	Secuencia vacía	List()
2	Seq(10)	Secuencia con un elemento	List(10)
3	Seq(20, 30, 10)	Mejor subsecuencia: 20,30	List(20, 30)
4	Seq(20, 30, 10, 40, 15, 16, 17)	Mejor subsecuencia: 10,15,16,17	List(10, 15, 16, 17)
5	Seq(10, 20, 30)	Secuencia completa es incremental	List(10, 20, 30)

## ssimlComenzandoEn

Caso	Entrada	Descripción	Resultado Esperado
1	(4, secuenciaLarga)	Comenzando en posición 4 (valor 6)	List(6, 22)
2	(12, secuenciaLarga)	Comenzando en posición 12 (valor 20)	List(20)
3	(0, Seq(10, 20, 30))	Comenzando en secuencia creciente	List(10, 20, 30)
4	(0, Seq(30, 20, 10))	Comenzando en secuencia decreciente	List(30)
5	(1, Seq(10, 20, 15))	Comenzando en posición 1 (valor 20)	List(20)

## subSecIncMasLargaV2

Caso	Entrada	Descripción	Resultado Esperado
1	Seq()	Secuencia vacía	List()
2	Seq(10)	Secuencia con un elemento	List(10)
3	Seq(20, 30, 10, 40, 15, 16, 17)	Secuencia compleja	List(10, 15, 16, 17)
4	secuenciaLarga	Secuencia larga con patrón	List(10, 22) o List(1, 22)
5	Seq(10, 20, 30)	Secuencia completamente creciente	List(10, 20, 30)

## Conclusiones sobre los Casos de Prueba

Los casos de prueba diseñados para las funciones implementadas en este taller han sido fundamentales para validar la corrección y robustez de las soluciones propuestas. Cada función fue sometida a una variedad de escenarios, incluyendo casos base, casos generales y casos límite, lo que permitió asegurar que las implementaciones cumplen con los requisitos especificados. Nos encontramos con algunos desafíos al diseñar casos que cubrieran todas las posibles variaciones de entrada, especialmente en funciones que manejan subsecuencias y combinaciones. Sin embargo, la exhaustividad de los casos de prueba nos permitió identificar y corregir errores potenciales, garantizando así la fiabilidad del código final.

## 4. Conclusiones

Tras la culminación de este taller, podemos extraer valiosas conclusiones sobre el proceso de resolución de problemas algorítmicos mediante programación funcional. La implementación del problema de la subsecuencia incremental más larga evidenció cómo el paradigma funcional facilita el desarrollo de soluciones robustas y elegantes. La inmutabilidad de los datos y la transparencia referencial de las funciones permitieron un razonamiento más claro sobre la corrección del código, mientras que la ausencia de efectos secundarios hizo que el comportamiento del programa fuera más predecible y fácil de verificar.

El uso sistemático de colecciones y expresiones `for` demostró ser una estrategia excepcionalmente efectiva para manipular datos de manera declarativa. En lugar de preocuparnos por los detalles de implementación de bucles y variables temporales, pudimos concentrarnos en la lógica fundamental del problema: generar combinaciones, filtrar elementos y transformar secuencias. Este enfoque no solo resultó en código más conciso, sino también más expresivo, donde la intención del programador se ve reflejada directamente en la estructura del código.