



Universidad del Valle
Facultad de ingeniería
Ingeniería en sistemas

Cristian David Pacheco Torres
2227437

Juan Sebastian Molina Cuellar
2224491

October 25, 2023

Taller 4: Colecciones y Expresiones For:
El problema de la subsecuencia incremental de longitud máxima

Contents

1	Vista general. Uso de colecciones y expresiones for.	3
2	Solución ingenua usando fuerza bruta	3
2.1	Generación de los índices asociados a todas las subsecuencias	3
2.1.1	Informe de uso de colecciones y expresiones for	3
2.1.2	Informe de corrección	4
2.1.3	Conclusiones	5
2.2	Generación de todas las subsecuencias de una secuencia	5
2.2.1	Informe de uso de colecciones y expresiones for	5
2.2.2	Informe de corrección	6
2.2.3	Conclusiones	8
2.3	Generación de todas las subsecuencias incrementales de una secuencia . .	8
2.3.1	Informe de uso de colecciones y expresiones for	8
2.3.2	Informe de corrección	9
2.3.3	Conclusiones	10
2.4	Hallar la subsecuencia incremental más larga	10
2.4.1	Informe de uso de colecciones y expresiones for	10
2.4.2	Informe de corrección	11
2.4.3	Conclusiones	12
3	Hacia una solución más eficiente	12
3.1	Calculando $SIML_i(S)$	12
3.1.1	Informe de uso de colecciones y expresiones for	12
3.1.2	Informe de corrección	12
3.1.3	Conclusiones	12
3.2	Calculando una subsecuencia incremental más larga, versión 2	12
3.2.1	Informe de uso de colecciones y expresiones for	12
3.2.2	Informe de corrección	12
3.2.3	Conclusiones	12

1 Vista general. Uso de colecciones y expresiones for.

El uso de colecciones y expresiones `for` en Scala representa una poderosa combinación que facilita la manipulación y transformación de datos. Las colecciones ofrecen una amplia gama de operaciones que permiten trabajar con conjuntos de datos de manera eficiente y expresiva. Por otro lado, las expresiones `for` proporcionan una sintaxis concisa para iterar y filtrar datos, haciendo que el código sea más legible y mantenible. Juntas, estas herramientas permiten a los desarrolladores escribir algoritmos complejos de manera más intuitiva, reduciendo la posibilidad de errores y mejorando la productividad.

2 Solución ingenua usando fuerza bruta

2.1 Generación de los índices asociados a todas las subsecuencias

2.1.1 Informe de uso de colecciones y expresiones for

```
1  def subindices(i: Int, n: Int): Set[Seq[Int]] = {  
2      val elements = (i until n).toSet  
3      (for {  
4          k <- 0 to elements.size  
5          combination <- elements.subsets(k)  
6      } yield combination.toSeq.sorted ).toSet  
7  }
```

Listing 1: Código en Scala para la función subindices

Tabla <i>subindices</i> (<i>i</i> : <i>Int</i> , <i>n</i> : <i>nt</i>)		
Función	¿Se utilizó colecciones y expresiones for?	¿Razón?
<i>subindices</i> (<i>i</i> : <i>Int</i> , <i>n</i> : <i>Int</i>)...	Sí	<p>Colecciones en Scala:</p> <ul style="list-style-type: none"> • Set y Seq permiten representar y manipular conjuntos y secuencias de datos de manera eficiente. • subsets(k) es una función de las colecciones que facilita la generación de todas las combinaciones posibles de un conjunto. • toSeq.sorted convierte un conjunto en una secuencia ordenada, lo cual es útil para garantizar la consistencia en las combinaciones generadas. <p>Expresiones for:</p> <ul style="list-style-type: none"> • Facilitan la iteración sobre colecciones y la generación de nuevas colecciones. • Permiten combinar múltiples generadores y filtros en una sola expresión, simplificando el código y haciéndolo más legible.

Table 1: Tabla de uso de colecciones y expresiones for en la función *subindices*(*i* : *Int*, *n* : *Int*)

2.1.2 Informe de corrección

Argumentación sobre la corrección:

Casos de prueba:

```

1 subindices(1, 5)
2 subindices(3, 6)
3 subindices(0, 4)
4 subindices(2, 7)
5 subindices(4, 8)

```

Listing 2: Casos de prueba para la función *subindices*

1. **Valor esperado:** HashSet(List(1), List(1, 2, 3), List(1, 3), List(3), List(), List(2, 3), List(1, 4), List(1, 3, 4), List(1, 2), List(2, 3, 4), List(3, 4), List(4), List(2), List(2, 4), List(1, 2, 3, 4), List(1, 2, 4))
2. **Valor esperado:** HashSet(List(4, 5), List(3), List(), List(3, 5), List(5), List(3, 4, 5), List(3, 4), List(4))
3. **Valor esperado:** HashSet(List(1, 2, 3), List(0, 1, 2, 3), List(0, 3), List(3), List(2, 3), List(0, 1), List(1, 2), List(0, 2), List(0), List(2), List(0, 1, 3), List(1), List(1, 3), List(0, 2, 3), List(0, 1, 2), List())
4. **Valor esperado:** HashSet(List(3, 5, 6), List(2, 3, 5, 6), List(5, 6), List(4, 5), List(), List(2, 3), List(2, 3, 5), List(3, 5), List(3, 6), List(3, 4), List(2, 3, 6), List(2), List(2,

4), List(4, 6), List(2, 4, 5), List(2, 3, 4, 5), List(2, 3, 4, 5, 6), List(3), List(3, 4, 5, 6), List(2, 3, 4, 6), List(2, 5, 6), List(4, 5, 6), List(5), List(6), List(2, 4, 6), List(3, 4, 6), List(3, 4, 5), List(2, 3, 4), List(2, 5), List(2, 6), List(2, 4, 5, 6), List(4))

5. **Valor esperado:** HashSet(List(4, 5, 7), List(4, 6, 7), List(5, 6), List(4, 5), List(), List(5, 7), List(4, 5, 6), List(6, 7), List(4), List(5, 6, 7), List(4, 7), List(4, 5, 6, 7), List(4, 6), List(7), List(5), List(6))

2.1.3 Conclusiones

2.2 Generación de todas las subsecuencias de una secuencia

2.2.1 Informe de uso de colecciones y expresiones for

```

1  def subSecuenciaAsoc(s:Secuencia, inds:Seq[Int]): Subsecuencia
    =
2    (for i <- 0 to inds.size-1 yield s(inds(i))).toList

```

Listing 3: Código en Scala para la funcion subSecuenciaAsoc

```

1  def subSecuenciasDe(s:Secuencia): Set[Subsecuencia] ={
2    val combinationIndex = subindices(0, s.size)
3    for index <- combinationIndex yield subSecuenciaAsoc(s, index
4    )
5  }

```

Listing 4: Código en Scala para la funcion subSecuenciasDe

Tabla <i>subSecuenciaAsoc</i> y <i>subSecuenciasDe</i>		
Función	¿Se utilizó colecciones y expresiones for?	¿Razón?
<i>subSecuenciaAsoc</i>	Sí	<p>Colecciones en Scala:</p> <ul style="list-style-type: none"> • Seq representa una secuencia de elementos en Scala. En esta función, se utiliza para representar una secuencia de índices. • toList convierte una colección en una lista. Esto puede ser útil para garantizar un tipo de salida específico o para realizar operaciones específicas de las listas. <p>Expresiones for:</p> <ul style="list-style-type: none"> • La expresión for se utiliza para iterar sobre la secuencia de índices y extraer los elementos correspondientes de la secuencia s. • Facilita la generación de una nueva colección basada en otra, en este caso, una subsecuencia basada en índices específicos.
<i>subSecuenciasDe</i>	Sí	<p>Uso de Funciones Preexistentes:</p> <ul style="list-style-type: none"> • La función subindices se utiliza para obtener todas las combinaciones posibles de índices para una secuencia dada. Esto demuestra la reutilización de código y la composición de funciones en Scala. <p>Generación de Subsecuencias:</p> <ul style="list-style-type: none"> • La expresión for se utiliza para iterar sobre cada combinación de índices y generar la subsecuencia correspondiente utilizando la función subSecuenciaAsoc.

Table 2: Tabla de uso de colecciones y expresiones for en la función *subSecuenciaAsoc* y *subSecuenciasDe*

2.2.2 Informe de corrección

Argumentación sobre la corrección:

Casos de prueba:

```

1  val s1 = Seq(5, 25, 35, 45, 55, 65, 75)
2  subSecuenciaAsoc(s1, Seq())
3  subSecuenciaAsoc(s1, Seq(0, 2, 4))
4  subSecuenciaAsoc(s1, Seq(1, 2, 4, 6))
5  subSecuenciaAsoc(s1, Seq(0, 3, 5))
6  subSecuenciaAsoc(s1, Seq(2, 3, 4, 5))

```

Listing 5: Casos de prueba para la función *subSecuenciaAsoc*

1. Valor esperado: `List()`

2. **Valor esperado:** List(5, 35, 55)
3. **Valor esperado:** List(25, 35, 55, 75)
4. **Valor esperado:** List(5, 45, 65)
5. **Valor esperado:** List(35, 45, 55, 65)

```

1  val s2 = Seq(20, 30, 10)
2  subSecuenciasDe(s2)
3  val s3 = Seq(10, 20)
4  subSecuenciasDe(s3)
5  val s4 = Seq(5, 15, 25, 35)
6  subSecuenciasDe(s4)
7  val s5 = Seq(1, 2, 3, 4, 5)
8  subSecuenciasDe(s5)
9  val s6 = Seq(50, 60, 70, 80, 90, 100)
10 subSecuenciasDe(s6)

```

Listing 6: Casos de prueba para la función subSecuenciasDe

1. **Valor esperado** (line 2): HashSet(List(30), List(20, 30, 10), List(30, 10), List(20), List(10), List(20, 30), List(20, 10), List())
2. **Valor esperado** (line 4): Set(List(), List(10), List(20), List(10, 20))
3. **Valor esperado** (line 6): HashSet(List(5, 15, 25, 35), List(), List(5, 15, 35), List(15, 35), List(5, 25, 35), List(15, 25, 35), List(25), List(35), List(15, 25), List(15), List(5, 35), List(5, 15), List(5), List(25, 35), List(5, 25), List(5, 15, 25))
4. **Valor esperado** (line 8): HashSet(List(1), List(1, 2, 3), List(1, 3), List(3, 4), List(4), List(2), List(1, 2, 3, 4), List(1, 2, 4), List(1, 2, 3, 4, 5), List(1, 2, 3, 5), List(1, 4, 5), List(4, 5), List(1, 3, 4, 5), List(2, 4, 5), List(2, 3, 4, 5), List(3), List(1, 5), List(), List(1, 2, 4, 5), List(2, 3), List(2, 3, 5), List(1, 4), List(1, 3, 4), List(3, 5), List(1, 2), List(1, 2, 5), List(5), List(3, 4, 5), List(2, 3, 4), List(2, 5), List(1, 3, 5), List(2, 4))
5. **Valor esperado** (line 10): HashSet(List(50, 60, 70), List(70), List(60, 80, 90), List(50, 60, 80, 90, 100), List(50, 70), List(60, 70, 80), List(100), List(50, 90, 100), List(50, 60, 90), List(90), List(60, 100), List(80, 90, 100), List(60, 70, 100), List(50, 80, 100), List(70, 100), List(50, 60, 70, 90, 100), List(50, 80, 90, 100), List(50, 60), List(50, 100), List(50, 70, 80, 90), List(50, 60, 100), List(50, 70, 90, 100), List(50, 60, 70, 80, 100), List(60, 70, 80, 90), List(60, 80), List(80, 100), List(70, 90, 100), List(50, 70, 80, 100), List(60, 70, 90, 100), List(60, 90), List(60, 80, 90, 100), List(50, 70, 100), List(80), List(60, 90, 100), List(), List(50, 60, 70, 90), List(50, 60, 70, 80, 90, 100), List(70, 80, 90, 100), List(50, 60, 80, 100), List(50, 60, 80), List(80, 90), List(60, 70, 80, 100), List(60, 70), List(50, 70, 80, 90, 100), List(90, 100), List(70, 80, 100), List(60, 80, 100), List(50, 70, 80), List(50, 80), List(60, 70, 90), List(50, 60, 70, 80, 90), List(70, 90), List(60, 70, 80, 90, 100), List(50, 80, 90), List(50), List(60), List(50, 60, 80, 90), List(70, 80), List(50, 90), List(50, 60, 70, 80), List(50, 60, 70, 100), List(50, 60, 90, 100), List(50, 70, 90), List(70, 80, 90))

2.2.3 Conclusiones

2.3 Generación de todas las subsecuencias incrementales de una secuencia

2.3.1 Informe de uso de colecciones y expresiones for

```
1  def incremental(seq: Subsecuencia): Boolean = seq match {  
2      case Nil => true  
3      case _ => (for index <- 1 to (seq.size - 1) yield seq(index  
4          - 1) < seq(index)) forall( x => x)  
}
```

Listing 7: Código en Scala para la funcion incremental

```
1  def subSecuenciasInc(seq: Secuencia): Set[Subsecuencia] =  
2      (for subsequence <- subSecuenciasDe(seq) if incremental(  
        subsequence) yield subsequence).toSet
```

Listing 8: Código en Scala para la funcion subSecuenciasInc

Tabla <i>incremental</i> y <i>subSecuenciasInc</i>		
Función	¿Se utilizó colecciones y expresiones for?	¿Razón?
<i>incremental</i>	Sí	<p>Pattern Matching:</p> <ul style="list-style-type: none"> El pattern matching es una característica poderosa de Scala que permite descomponer y verificar estructuras de datos. En este caso, se utiliza para manejar dos escenarios: cuando la subsecuencia es vacía (representada por 'Nil') y cuando no lo es. <p>Verificación Incremental:</p> <ul style="list-style-type: none"> La expresión <code>for</code> se utiliza para iterar sobre la subsecuencia y verificar si cada elemento es menor que el siguiente. Esto genera una colección de valores booleanos. La función <code>forall</code> se utiliza para verificar que todos los valores en la colección booleana sean <code>true</code>, lo que indica que la subsecuencia es incremental.
<i>subSecuenciasInc</i>	Sí	<p>Generación y Filtrado de Subsecuencias:</p> <ul style="list-style-type: none"> La función <code>subSecuenciasDe</code> se utiliza para generar todas las posibles subsecuencias de la secuencia dada. La cláusula <code>if incremental(subsequence)</code> dentro de la expresión <code>for</code> filtra las subsecuencias, conservando solo aquellas que son incrementales. Esto demuestra cómo Scala permite combinar la generación y el filtrado de colecciones de manera concisa. <p>Conversión a Conjunto:</p> <ul style="list-style-type: none"> <code>toSet</code> convierte la colección resultante en un conjunto, eliminando posibles duplicados y garantizando la unicidad de las subsecuencias.

Table 3: Tabla de uso de colecciones y expresiones for en la función *incremental* y *subSecuenciasInc*

2.3.2 Informe de corrección

Argumentación sobre la corrección:

Casos de prueba:

```

1  val s7 = Seq(1, 2, 3, 4, 5, 6, 7)
2  incremental(s7) // true
3  val s8 = Seq()
4  incremental(s8) // true
5  val s9 = Seq(1, 1, 1, 1, 1, 1, 1)

```

```

6 incremental(s9)//false
7 val s10 = Seq(1, 2, 3, 5, 4, 6, 7)
8 incremental(s10)//false
9 val s11 = Seq(7, 6, 5, 4, 3, 2, 1)
10 incremental(s11)//false

```

Listing 9: Casos de prueba para la función incremental

```

1 subSecuenciasInc(Seq(1, 2))
2 subSecuenciasInc(Seq(5, 7, 9))
3 subSecuenciasInc(Seq(2, 4, 8, 16))
4 subSecuenciasInc(Seq(0, 1, 2))
5 subSecuenciasInc(Seq(10, 20, 30, 40))

```

Listing 10: Casos de prueba para la función subSecuenciasInc

1. **Valor esperado:** Set(List(), List(1), List(2), List(1, 2))
2. **Valor esperado:** HashSet(List(5, 9), List(9), List(7), List(), List(5, 7), List(7, 9), List(5, 7, 9), List(5))
3. **Valor esperado:** HashSet(List(8), List(16), List(4, 16), List(2, 8), List(2, 4, 16), List(4), List(2, 16), List(2, 8, 16), List(8, 16), List(4, 8, 16), List(), List(2), List(2, 4, 8, 16), List(4, 8), List(2, 4), List(2, 4, 8))
4. **Valor esperado:** HashSet(List(1), List(0, 1), List(1, 2), List(0, 2), List(0), List(2), List(0, 1, 2), List())
5. **Valor esperado:** HashSet(List(30, 40), List(10, 20), List(10, 40), List(20, 40), List(30), List(10, 30, 40), List(10), List(20, 30, 40), List(20, 30), List(10, 20, 30), List(10, 30), List(), List(20), List(40), List(10, 20, 40), List(10, 20, 30, 40))

2.3.3 Conclusiones

2.4 Hallar la subsecuencia incremental más larga

2.4.1 Informe de uso de colecciones y expresiones for

```

1 def subsecuenciaIncrementalMasLarga(seq: Secuencia): Subsecuencia
  = {
2   val subsequences = (for subsequence <- subSecuenciasInc(seq) if
    incremental(subsequence) yield subsequence).toList
3   val subsequencesSizes = subsequences.map(_.size)
4   val indexOfLargestSubsequence = (subsequencesSizes.indexOf(x
    => x == subsequencesSizes.max))
5   indexOfLargestSubsequence match {
6     case x if x < 0 => List()
7     case x => subsequences(x)
8   }
9 }

```

Listing 11: Código en Scala para la función subsecuenciaIncrementalMasLarga

Tabla base <i>subsecuenciaIncrementalMasLarga</i>		
Función	¿Se utilizó colecciones y expresiones for?	¿Razón?
subsecuenciaIncrementalMasLarga	Sí	<p>Generación y Filtrado de Subsecuencias Incrementales:</p> <ul style="list-style-type: none"> Se utiliza la función <code>subSecuenciasInc</code> para obtener todas las subsecuencias incrementales de la secuencia dada. Aunque <code>subSecuenciasInc</code> ya filtra las subsecuencias incrementales, la cláusula <code>if incremental(subsequence)</code> se mantiene por claridad y robustez. <p>Determinación de la Subsecuencia Más Larga:</p> <ul style="list-style-type: none"> <code>map</code> se utiliza para transformar la lista de subsecuencias en una lista de sus tamaños. <code>find</code> se utiliza para obtener el índice de la subsecuencia más larga. <p>Pattern Matching para la Salida:</p> <ul style="list-style-type: none"> Se utiliza pattern matching para manejar diferentes escenarios: cuando no se encuentra ninguna subsecuencia, cuando se encuentra una subsecuencia y cualquier otro caso.

Table 4: Tabla de uso de colecciones y expresiones for en la función *subsecuenciaIncrementalMasLarga*

2.4.2 Informe de corrección

Argumentación sobre la corrección:

Casos de prueba:

```

1  val s14 = Seq(1, 2, 3, 4, 5)
2  subsecuenciaIncrementalMasLarga(s14) //List(1, 2, 3, 4, 5)
3  val s15 = Seq(5, 10, 15, 14, 13, 12)
4  subsecuenciaIncrementalMasLarga(s15) //List(5, 10, 14)
5  val s16 = Seq(2, 4, 8, 7, 6, 5)
6  subsecuenciaIncrementalMasLarga(s16) //List(2, 4, 7)
7  val s17 = Seq(0, 1, 1, 2, 3, 5, 4)
8  subsecuenciaIncrementalMasLarga(s17) //List(0, 1, 2, 3, 5)
9  val s18 = Seq(10, 20, 30, 25, 35, 45)
10 subsecuenciaIncrementalMasLarga(s18) //List(10, 20, 30, 35, 45)

```

Listing 12: Casos de prueba para la función *subsecuenciaIncrementalMasLarga*

2.4.3 Conclusiones

3 Hacia una solución más eficiente

3.1 Calculando $SIML_i(S)$

3.1.1 Informe de uso de colecciones y expresiones for

Tabla base completar		
Función	¿Se utilizó colecciones y expresiones for?	¿Razón?
Completar...	Completar...	Completar...

Table 5: Completar...

3.1.2 Informe de corrección

Argumentación sobre la corrección:

Casos de prueba:

3.1.3 Conclusiones

3.2 Calculando una subsecuencia incremental más larga, versión 2

3.2.1 Informe de uso de colecciones y expresiones for

Tabla base completar		
Función	¿Se utilizó colecciones y expresiones for?	¿Razón?
Completar...	Completar...	Completar...

Table 6: Completar...

3.2.2 Informe de corrección

Argumentación sobre la corrección:

Casos de prueba:

3.2.3 Conclusiones