Taller 4: Colecciones y Expresiones For

Juan Francisco Díaz Frias Samuel Galindo

Octubre 2023

1. El problema de la subsecuencia incremental de longitud máxima

Dada una secuencia de elementos, $S =< s_1, s_2, \ldots, s_n >$, una subsecuencia de S es una secuencia obtenida a partir de S borrando 0 o más elementos sin cambiar el orden de los que quedan. Formalmente, una subsecuencia de longitud m de S es una secuencia de la forma $< s_{i_1}, s_{i_2}, \ldots, s_{i_m} >$ donde $1 \leq i_1 < i_2 < \ldots < i_m \leq n$.

Las secuencias son una colección muy utilizada para modelar y resolver problemas cotidianos, y en muchos de ellos encontrar subsecuencias es muy útil.

Sea ss una subsecuencia de longitud m de S $ss =< s_{i_1}, s_{i_2}, \ldots, s_{i_m} >$ donde $1 \leq i_1 < i_2 < \ldots < i_m \leq n$. Se dice que ss es incremental, si $s_{i_1} < s_{i_2} < \ldots < s_{i_m}$.

Por ejemplo, si $S =< 20, 30, 10, 40, 15, 16, 17, 11, 12, 13, 14 >$ las subsecuencias $< 30, 40 >, < 10, 15, 17 >$ son subsecuencias incrementales, pero las subsecuencias $< 30, 10, 40 >, < 20, 40, 15 >, < 10, 16, 17, 11, 14 >$ no lo son. La subsecuencia incremental más larga de S es $< 10, 11, 12, 13, 14 >$ (¿Cuáles son las otras subsecuencias incrementales? ¿Cuántas hay en total?)

En este ejercicio de programación, el objetivo es encontrar la subsecuencia incremental más larga de una secuencia de números enteros:

El problema de la subsecuencia incremental más larga (SIML)

Entrada: $S =< s_1, s_2, \ldots, s_n >, s_i \in (N)$

Salida: $ss =< s_{i_1}, s_{i_2}, \ldots, s_{i_m} >$ subsecuencia incremental de S tal que es la más larga posible

Representación de los datos Para modelar un a secuencia y una subsecuencia de enteros se tienen los siguientes tipos de datos:

```
type Secuencia = Seq[Int]
type Subsecuencia = Seq[Int]
```

1.1. Solución ingenua usando fuerza bruta

La primera solución que usted debe programar para resolver este problema consiste en generar todas las subsecuencias incrementales de una secuencia S, y entre ellas escoger la más larga.

Para ello usted debe implementar las siguientes funciones.

1.1.1. Generación de los índices asociados a todas las subsecuencias

Tal como se dijo anteriormente, formalmente, una subsecuencia de longitud m de S es una secuencia de la forma $< s_{i_1}, s_{i_2}, \ldots, s_{i_m} >$ donde $1 \leq i_1 < i_2 < \ldots < i_m \leq n$.

Eso quiere decir que toda secuencia de índices que cumpla estas condiciones, representa una posible subsecuencia. Su tarea en este apartado es implementar la función `subindices` que dados i y n, con $i \leq n$, devuelve el **conjunto** de todas las posibles secuencias crecientes que se pueden hacer con los índices $i..n$.

```
def subindices(i:Int, n:Int): Set[Seq[Int]] = {
  // dados i y  n devuelve todas las posibles secuencias crecientes de enteros entre i y n
...
}
```

Nótese que la secuencia vacía de índices siempre es una solución. También note que si n es la longitud de una secuencia, los índices de las subsecuencias están entre 0 y $n-1$.

Un ejemplo de uso y respuesta de la función es el siguiente:

```
scala> subindices(2,3)
val res12: Set[Seq[Int]] = Set(List(), List(2))

scala> subindices(0,3)
val res13: Set[Seq[Int]] = HashSet(List(1), List(0, 1), List(1, 2), List(0, 2), List(0),
                                   List(2), List(0, 1, 2), List())
```

1.1.2. Generación de todas las subsecuencias de una secuencia

Implemente la función `subSecuenciaAsoc` que dada una secuencia s y una secuencia creciente de índices *inds* asociada a s, devuelva la subsecuencia correspondiente a ese conjunto de índices.

```
def subSecuenciaAsoc(s:Secuencia,inds:Seq[Int]): Subsecuencia = {
  // Dadas s, una secuencia, e inds, una secuencia creciente de indices asociada a s,
  // Devuelve la secuencia correspondiente a la secuencia creciente de indices inds
...
}
```

Un ejemplo de uso y respuesta de la función es el siguiente:

```
scala> val s = Seq(20,30,10,40,15,16,17)
val s: Seq[Int] = List(20, 30, 10, 40, 15, 16, 17)
```

```
scala> subSecuenciaAsoc(s, Seq())
val res0: SubsecuenciaMasLarga.Subsecuencia = List()

scala> subSecuenciaAsoc(s, Seq(0,2,4))
val res1: SubsecuenciaMasLarga.Subsecuencia = List(20, 10, 15)

scala> subSecuenciaAsoc(s, Seq(1,2,4,6))
val res2: SubsecuenciaMasLarga.Subsecuencia = List(30, 10, 15, 17)
```

Usando esta función, implemente la función `subSecuenciasDe` que dada una secuencia s devuelva el conjunto de todas las subsecuencias posibles de s (¿Cuántas hay?).

```
  def subSecuenciasDe(s:Secuencia): Set[Subsecuencia] = {
    // Dada s, devuelve el conjunto de todas las subsecuencias posibles de s
...
  }
```

Un ejemplo de uso y respuesta de la función es el siguiente:

```
scala> val s1=Seq(20,30,10)
val s1: Seq[Int] = List(20, 30, 10)

scala> subSecuenciasDe(s1)
val res7: Set[SubsecuenciaMasLarga.Subsecuencia] = HashSet(List(30), List(20, 30, 10), List(30, 10),
                                                List(20), List(10), List(20, 30), List(20, 10), List())
```

1.1.3. Generación de todas las subsecuencias incrementales de una secuencia

Implemente la función booleana `incremental` que dada una subsecuencia decida si es incremental (devuelve true) o no (devuelve false).

```
  def incremental(s:Subsecuencia):Boolean = {
...
  }
```

Ahora implemente la función `subSecuenciasInc` que dada una secuencia s devuelva el **conjunto** de todas las subsecuencias incrementales de s.

```
  def subSecuenciasInc(s:Secuencia): Set[Subsecuencia] = {
...
  }
```

Un ejemplo de uso y respuesta de la función es el siguiente:

```
scala> s
val res10: Seq[Int] = List(20, 30, 10, 40, 15, 16, 17)

scala> s1
val res11: Seq[Int] = List(20, 30, 10)

scala> subSecuenciasInc(s)
val res12: Set[SubsecuenciaMasLarga.Subsecuencia] = HashSet(List(16, 17), List(17), List(30, 40),
List(10, 40),List(16), List(20, 40), List(15), List(), List(10, 15, 16), List(15, 16, 17), List(10, 17),
List(10, 16), List(10, 15, 17), List(15, 17), List(20, 30, 40), List(20, 30), List(10, 15),
List(10, 15, 16, 17), List(30), List(15, 16), List(20), List(40), List(10), List(10, 16, 17))

scala> subSecuenciasInc(s1)
val res13: Set[SubsecuenciaMasLarga.Subsecuencia] = HashSet(List(30), List(), List(20),
                                                List(10), List(20, 30))
```

1.1.4. Hallar la subsecuencia incremental más larga

Utilizando la función anterior, implemente la función `subsecuenciaIncrementalMasLarga` que recibe una secuencia s y devuelve una subsecuencia incremental de tamaño máximo.

```
  def subsecuenciaIncrementalMasLarga(s:Secuencia): Subsecuencia = {
...
  }
```

Un ejemplo de uso y respuesta de la función es el siguiente:

```
scala> s
val res16: Seq[Int] = List(20, 30, 10, 40, 15, 16, 17)

scala> s1
val res17: Seq[Int] = List(20, 30, 10)

scala> subsecuenciaIncrementalMasLarga(s)
val res18: SubsecuenciaMasLarga.Subsecuencia = List(10, 15, 16, 17)

scala> subsecuenciaIncrementalMasLarga(s1)
val res19: SubsecuenciaMasLarga.Subsecuencia = List(20, 30)
```

1.2. Hacia una solución más eficiente

Aunque tenemos una solución al problema, esta es bastante ineficiente, pues genera inicialmente todas las subsecuencias posibles.

Sin embargo, estudiando el problema nos podemos aprovechar de una propiedad para diseñar un algoritmo más eficiente: Nótese que si $< s_{i_1}, s_{i_2}, \ldots, s_{i_m} >$ es la subsecuencia incremental más larga de $S =< s_1, s_2, \ldots, s_n >$, entonces $< s_{i_2}, \ldots, s_{i_m} >$ es la subsecuencia incremental más larga de $S_{i_2} =< s_{i_2}, s_{i_2+1}, s_{i_2+2}, \ldots, s_n >$ (¿podría usted argumentar por qué esta observación es cierta?).

Eso quiere decir que encontrar la solución al problema original (hallar la subsecuencia incremental más larga de S) puede ser expresado en función de encontrar soluciones a problemas más pequeños que el original. Para entender la idea vamos a llamar S_i a la subsecuencia completa de S que comienza en s_i. Es decir, $S_i =< s_i, s_{i+1}, s_{i+2}, \ldots, s_n >$ (nótese que $S_1 = S$).

Ahora note que si la subsecuencia incremental más larga de S es $< s_{i_1}, s_{i_2}, \ldots, s_{i_m} >$, se tiene que:

- $i_1 \in 1..n$
- i_2 es tal que $s_{i_2} > s_{i_1}$

Es decir, el primer elemento de la subsecuencia puede ser cualquiera (no sabemos cuál), pero el segundo tiene que ser uno mayor que el primer elemento. Y si hay varios mayores que el primero, el segundo tiene que ser uno de ellos.

Por tanto, se puede concluir que:

- Para hallar la subsecuencia incremental más larga de S , es suficiente hallar la subsecuencia incremental más larga que comienza en s_1, luego la más larga que

comienza en s_2 y así sucesivamente hasta hallar la más larga que comienza en s_n, y entre todas ellas escoger la más larga.

- Hallar la subsecuencia incremental más larga de S que comienza en s_i, es simplemente hallar las subsecuencias incrementales más largas de S que comienzan en s_j para los $j : s_j > s_i$, escoger entre ellas la más larga, y anteponerle s_i a la secuencia.

Más formalmente, sea $SIML_i(S)$ la subsecuencia incremental más larga de S que comienza en s_i. Las siguientes ecuaciones son ciertas:

- $SIML(S) = max\{SIML_i(S) : i \in 1..n\}$
- $SIML_i(S) = s_i :: max\{SIML_j(S) : j > i \wedge s_j > s_i\}$

1.2.1. Calculando $SIML_i(S)$

Implemente la función `ssimlComenzandoEn` que dados un índice i y una secuencia s devuelva la subsecuencia incremental más larga de s que comienza en $s(i)$.

```
  def ssimlComenzandoEn(i:Int, s:Secuencia):Subsecuencia = {
    // Devuelve la subsecuencia incremental mas larga de s que comienza en s(i)
...
  }
```

Un ejemplo de uso y respuesta de la función es el siguiente:

```
scala> ssimlComenzandoEn(4, Seq(10, 9, 8, 7, 6, 5, 4, 3, 2,1, 22,21,20,19,18,17,16,15,14,13,12,11))
val res7: SubsecuenciaMasLarga.Subsecuencia = List(6, 22)

scala> ssimlComenzandoEn(12, Seq(10, 9, 8, 7, 6, 5, 4, 3, 2,1, 22,21,20,19,18,17,16,15,14,13,12,11))
val res8: SubsecuenciaMasLarga.Subsecuencia = List(20)
```

1.2.2. Calculando una subsecuencia incremental más larga, versión 2

Utilizando la función anterior, implemente la función `subSecIncMasLargaV2` que recibe una secuencia s y devuelve una subsecuencia incremental de tamaño máximo.

```
  def subSecIncMasLargaV2(s:Secuencia): Subsecuencia = {
...
  }
```

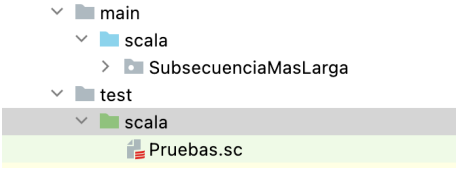
Un ejemplo de uso y respuesta de la función es el siguiente:

```
scala> subSecIncMasLargaV2(Seq(10, 9, 8, 7, 6, 5, 4, 3, 2,1, 22,21,20,19,18,17,16,15,14,13,12,11))
val res6: SubsecuenciaMasLarga.Subsecuencia = List(10, 22)
```

2. Entrega

2.1. Paquete *SubsecuenciaMasLarga* y *worksheet* de pruebas

Usted deberá entregar dos archivos `package.scala` y `pruebas.sc` los cuales harán parte de su estructura de proyecto `IntellijIdea`, como se muestra en la figura a continuación:



Las funciones correspondientes a cada ejercicio, `subindices`, `subSecuenciaAsoc`, `subSecuenciasDe`, `incremental`, `subSecuenciasInc`, `subsecuenciaIncrementalMasLarga`, `ssimlComenzandoEn` y `subSecIncMasLargaV2` deben ser implementadas en un paquete de Scala denominado `SubsecuenciaMasLarga`. **Utilice expresiones for para implementar todas la funciones**. En ese paquete debe venir un archivo denominado `package.scala` que debe tener la forma siguiente:

```
package object SubsecuenciaMasLarga {
  type Secuencia = Seq[Int]
  type Subsecuencia = Seq[Int]
  // Una subsecuencia es definida por  una secuencia creciente de enteros que representan
  // los indices de los elementos en la secuencia que hacen parte de la subsecuencia
  // si Seq(i1,i2,i3), con i1<i2<i3 es una secuencia creciente de enteros en [0..s.lenght-1]
  // esta secuencia de indices representa  la subsecuencia Seq(s[i1], s[i2], s[i3]) de s
...
  }

  def subSecuenciaAsoc(s:Secuencia,inds:Seq[Int]): Subsecuencia = {
    // Dadas s, una secuencia, e inds, una secuencia creciente de indices asociada a s,
    // Devuelve la secuencia correspondiente a la secuencia creciente de indices inds
...
  }

  def subSecuenciasDe(s:Secuencia): Set[Subsecuencia] = {
    // Dada s, devuelve el conjunto de todas las subsecuencias posibles de s
...
  }

  def incremental(s:Subsecuencia):Boolean = {
...
  }

  def subSecuenciasInc(s:Secuencia): Set[Subsecuencia] = {
...
  }

  def subsecuenciaIncrementalMasLarga(s:Secuencia): Subsecuencia = {
...
  }


  def ssimlComenzandoEn(i:Int, s:Secuencia):Subsecuencia = {
    // Devuelve la subsecuencia incremental mas larga de s que comienza en s(i)
...
  }

  def subSecIncMasLargaV2(s:Secuencia): Subsecuencia ={
    // Devuelve una subsecuencia incremental mas larga de s
...
}
```

Dicho paquete será usado en un *worksheet* de Scala con casos de prueba. Estos casos de prueba deben venir en un archivo denominado `pruebas.sc` . Un ejemplo de un tal archivo es el siguiente:

```
import SubsecuenciaMasLarga._
subindices(2,3)
subindices(0,3)
subSecuenciasDe(Seq(20,30,10))
subSecuenciasInc(Seq(20,30,10))
subSecuenciasInc(Seq(10,20,30))
subsecuenciaIncrementalMasLarga(Seq(20,30,10,40,15,16,17))
subSecIncMasLargaV2(Seq(20,30,10,40,15,16,17))
subsecuenciaIncrementalMasLarga(Seq(20,30,10,40,15,16,17,11, 12, 13,14))
subSecIncMasLargaV2(Seq(20,30,10,40,15,16,17,11, 12, 13,14))
```

2.2. Informe del taller - secciones

Todo taller debe venir acompañado de un informe en formato pdf. El informe debe contener la información explícita solicitada en el enunciado.

Para este caso, el informe del taller debe contener al menos tres secciones: informe de uso de colecciones y expresiones for, informe de corrección y conclusiones.

2.2.1. Informe de uso de colecciones y expresiones for

Tal como se ha visto en clase, el uso de colecciones y expresiones for es una herramienta muy poderosa y expresiva para programar. En esta sección usted debe hacer una tabla indicando en cuáles funciones usó la técnica y en cuáles no. Y en las que no, indicar por qué no lo hizo.

También se espera una apreciación corta de su parte, sobre el uso de colecciones y expresiones for como técnica de programación.

2.2.2. Informe de corrección

Es muy importante reflexionar sobre la corrección del código entregado. Para ello se deberá argumentar sobre la corrección de las funciones entregadas, y también deberá entregar un conjunto de pruebas. Todo esto lo consigna en esta sección del informe, dividida de la siguiente manera:

Argumentación sobre la corrección .

Para cada función, `subindices`, `subSecuenciaAsoc`, `subSecuenciasDe`, `incremental`, `subSecuenciasInc`, `subsecuenciaIncrementalMasLarga`, `ssimlComenzandoEn` y `subSecIncMasLargaV2` argumente lo más formalmente posible por qué es correcta. Utilice inducción o inducción estructural donde lo vea pertinente. Estas argumentaciones las consigna en esta sección del informe.

Casos de prueba .

Para cada función se requieren mínimo 5 casos de prueba donde se conozca claramente el valor esperado, y se pueda evidenciar que el valor calculado por la función corresponde con el valor esperado en toda ocasión.

Una descripción de los casos de prueba diseñados, sus resultados y una argumentación de si cree o no que los casos de prueba son suficientes para confiar en la corrección de cada uno de sus programas, los registra en esta sección del informe. Obviamente, esta parte del informe debe ser coherente con el archivo de pruebas entregado, `pruebas.sc`.

2.3. Fecha y medio de entrega

Todo lo anterior, es decir los archivos `package.scala`, `pruebas.sc`, e `Informe del taller`, debe ser entregado vía el campus virtual, a más tardar a las 10 a.m. del jueves 26 de octubre de 2023, en un archivo comprimido que traiga estos tres elementos.

Cualquier indicación adicional será informada vía el foro del campus asociado a *programación funcional*.

3. Evaluación

Cada taller será evaluado tanto por el profesor del curso con ayuda del monitor, como por 3 compañeros que hayan entregado el taller. A este tipo de evaluación se le conoce como *Coevaluación*.

El objetivo de la coevaluación es lograr aprendizajes a través de:

- La lectura de lo que otros compañeros hicieron ante el mismo reto. Esto permite contrastar las soluciones propias con las de otros, y aprender de ellas, o compartir mejores maneras de hacer algo con otros.
- Retroalimentar a los compañeros que fueron asignados para evaluar. Al escribir la percepción que tenemos sobre el trabajo del otro, podemos aprender de cómo lo hicieron, y dar indicaciones al otro sobre otra formas de hacer lo mismo o, incluso, felicitarlo por la solución que presenta.
- La lectura de las retroalimentaciones de mis compañeros o del profesor/monitor.

La calificación de cada taller corresponderá entonces:

- en un 80 % a la calificación ponderada que reciba del profesor/monitor, vía una rúbrica de evaluación (pesa 5 veces lo que pesa la de otro estudiante) y de los tres compañeros asignados para evaluarlo.
- en un 20 % a la calificación que el sistema hará del trabajo de evaluación asignado. El Sistema tiene un método inteligente para estimar esa calificación, a partir de las evaluaciones realizadas por cada estudiante y por el profesor/monitor.