Taller 1: Recursión

Juan Francisco Díaz Frias Santiago Casañas

Septiembre 2022

1. Ejercicios de programación

1.1. Máximo común divisor

Recordemos del curso de matemáticas discretas la definición del máximo común divisor. Sean $n, m \in \mathbb{N} \setminus \{0\}$. El máximo común divisor de n y m, $mcd(n, m)$, es el número natural d más grande que divide tanto a n como a m. Formalmente:

$mcd : \mathbb{N} \setminus \{0\} \times \mathbb{N} \setminus \{0\} \to \mathbb{N}$ tal que:

$$mcd(n, m) = (max\ d \in \mathbb{N} |\ \ d|n \wedge d|m : d)$$

En este taller usted deberá implementar varias versiones de este algoritmo de forma recursiva.

1.1.1. Utilizando el Teorema Fundamental de la Aritmética

Teorema fundamental de la aritmética: Todo número natural mayor que 1, se puede expresar de manera única como producto de números primos. Si los primos se presentan en orden ascendente, esta descomposición es única.

$$\forall n \in \mathbb{N} \exists k \in \mathbb{N} \exists p_1, p_2, \ldots, p_k \in \mathbb{N} | primo(p_1) \wedge \ldots \wedge primo(p_k) :$$

$$(p_1 \leq p_2 \leq \ldots \leq p_k) \wedge (n = p_1 p_2 \ldots p_k)$$

Por ejemplo:

- $100 = 2 * 2 * 5 * 5 = 2^2 * 5^2$
- $641 = 641$

- $999 = 3 * 3 * 3 * 37 = 3^3 * 37$
- $1024 = 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 = 2^{10}$

Por el teorema fundamental de la aritmética, podemos escribir cualquier n como:

$$(n = p_1^{i_1} p_2^{i_2} \dots p_k^{i_k}) \wedge (p_1 < p_2 < \dots < p_k) \wedge (\forall j| : k_j > 0)$$

Por ejemplo, $120 = 2^3 * 3 * 5$ y $500 = 2^2 * 5^3$

Nótese que $3 \not| 500$, pero podríamos escribir $500 = 2^2 * 3^0 * 5^3$, y así usaríamos los mismos primos en ambas descomposiciones

Entonces, sin pérdida de generalidad podemos escribir:

$$(n = p_1^{i_1} p_2^{i_2} \dots p_k^{i_k}) \wedge (\forall j | 1 \leq j \leq k : i_j \geq 0)$$

donde agregamos primos elevados a la cero si los necesitamos:

$120 = 2^3 * 3^1 * 5^1$ no agregamos

$500 = 2^2 * 3^0 * 5^3$ si agregamos

Entonces, dados:

$$n = p_1^{i_1} p_2^{i_2} \dots p_k^{i_k} \text{ y } m = p_1^{j_1} p_2^{j_2} \dots p_k^{j_k}$$

se tiene que:

$$mcd(n, m) = p_1^{min(i_1, j_1)} p_2^{min(i_2, j_2)} \dots p_k^{min(i_k, j_k)}$$

Por tanto,

$$mcd(120, 500) = 2^2 * 3^0 * 5^1 = 20$$

Implemente la función `mcdTFA`:

```
def mcdTFA(ln:List[Int], lm:List[Int], primos:List[Int]) : Int =  ...
```

que recibe la descomposición en primos de $n, m \in \mathbb{N} \setminus \{0\}$, en tres listas $ln, lm, primos$, donde

- *primos* es la lista ordenada de primos usados en las dos representaciones
- ln es la lista de exponentes de los primos de *primos* en la representación de n, y
- lm es la lista de exponentes de los primos de *primos* en la representación de m.

y devuelve el $mcd(n.m)$. Por ejemplo:

```
mcdTFA(List(3,1,1),List(2,0,3),List(2,3,5))
```

calcula el $mcd(120, 500)$.

1.1.2. Utilizando el algoritmo de la división

El ejercicio anterior necesita la descomposición en primos de n y m para poder calcular su máximo común divisor. Usando el algoritmo de la división, esta tarea puede ser más sencilla.

El algoritmo de la división establece que: Dados $n \in \mathbb{N}, d \in \mathbb{N}^+$.

$$\exists q, r | 0 \leq r < d : n = qd + r$$

y q y r son únicos

Basados en este hecho y en la siguiente propiedad, (suponemos $n \geq m$):

$$n = mq + r \implies mcd(n, m) = mcd(m, r)$$

se tiene que:

$$mcd(n, m) = \begin{cases} n & \text{Si } m = 0 \\ mcd(m, r) & \text{Si, por algoritmo de la división } n = mq + r \end{cases}$$

Por ejemplo, calculemos $mcd(963, 657)$:

Paso	n	m	q	r
0	963	657	1	306
1	657	306	2	45
2	306	45	6	36
3	45	36	1	9
4	36	9	4	0
5	9	0		

Una de la propiedades del mcd es:

$$mcd(m, n) = d \implies \exists x, y | : d = mx + ny$$

El algoritmo de Euclides también permite calcular x y y tal que $mcd(n, m) = xm + yn$
Por ejemplo, a partir de la tabla resultante anterior:

Del Paso	$r =$
3	$9 = 45 - 36 * 1$
2	$36 = 306 - 45 * 6$
1	$45 = 657 - 306 * 2$
0	$306 = 963 - 657 * 1$

$$9 = 45 - 36 * 1 = -306 + 45 * 7 = 657 * 7 - 306 * 15 = 963 * (-15) + 657 * 22$$

Implemente la función `mcdEB`:

```
def mcdEB(n:Int, m:Int):(Int, Int, Int) = ...
```

que recibe $n, m \in \mathbb{N} \setminus \{0\} \wedge n \geq m$, y devuelve una tripleta (d, x, y) tal que

- $d = mcd(n, m)$
- $d = n * x + m * y$

Por ejemplo:

```
mcdEB(963, 657)
```

devuelve $(9, -15, 22)$
Hay tres métodos que se proveen en List[Int] que pueden ser útiles para este ejercicio:

- l.isEmpty: Boolean (devuelve si una lista l está vacía)
- l.head: Int (devuelve el primer elemento de la lista l)
- l.tail: List[Int] (devuelve la lista sin el primer elemento l)

1.2. Números de Fibonacci

Una pareja de conejos recién nacidos (macho y hembra) es dejada en una isla. Se sabe que los conejos no se pueden reproducir hasta que no cumplen dos meses de edad. Una vez una pareja de conejos cumple 2 meses de edad, se reproducen y dan a luz una pareja nueva. Se quiere definir una función $fib : \mathbb{N} \to \mathbb{N}$ tal que $fib(n)$ represente el número de parejas de conejos en la isla después de pasados n meses. Suponga que ninguna pareja de conejos se muere.

Miremos el comportamiento en una tabla:

Mes	Edad			Total
	≥ 2	1	0	
0	0	0	1	1
1	0	1	0	1
2	1	0	1	2
3	1	1	1	3
4	2	1	2	5
5	3	2	3	8
6	5	3	5	13

Implemente las funciones `fibonacciA`, y `fibonacciI` que calculen $fib(n)$, pero la primera genere un proceso de recursión de árbol, mientras la segunda genere un proceso iterativo.

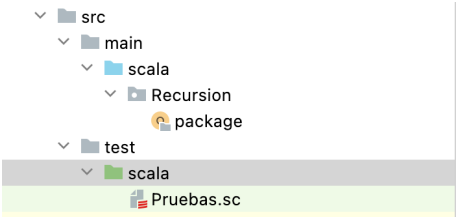
```
  def fibonacci(n:Int):Int = ...

  def fib(n:Int):Int = ...
```

2. Entrega

2.1. Paquete *Recursion* 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 4 funciones correspondientes a cada ejercicio, `mcdTFA`, `mcdEB`, `fibonacciA`, y `fibonacciI` deben ser implementadas en un paquete de Scala denominado `Recursion`. En ese paquete debe venir un archivo denominado `package.scala` que debe tener la forma siguiente:

```
package object Recursion {
   /**
   * Ejercicio 1.1.1
   * maximo comun divisor a partir del teorema fundamental de la aritmetica
   */
  def mcdTFA(ln:List[Int], lm:List[Int], primos:List[Int]) : Int = ...

  /**
   * Ejercicio 1.1.2
   * maximo comun divisor a partir del teorema de Euclides con coeficientes de Bezout
   */

  def mcdEBez(n:Int, m:Int):(Int,Int,Int) =  ...

  /**
   * Ejercicio 1.2.1
   * fibonacci recursivo de arbol
   */
  def fibonacciA(n:Int):Int = ...

  /**
   * Ejercicio 1.2.2
   * fibonacci iterativo
   */

  def fibonacciI(n:Int):Int = ...
}
```

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 Recursion.{mcdTFA, mcdEBez, fibonacciA,fibonacciI}

//Pruebas mcdTFA

mcdTFA(List(3,1,1),List(2,0,3),List(2,3,5)) // mcd(120,500)
...

//Pruebas mcdEBez

mcdEBez(120,500)
...

//Pruebas fibonacciA

fibonacciA(5)
...

//Pruebas fibonacciI

fibonacciI(5)
...
```

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 procesos, informe de corrección y conclusiones.

2.2.1. Informe de procesos

Tal como se ha visto en clase, los procesos generados por los programas recursivos podrían ser iterativos o recursivos (lineales o de árbol). Para cada una de las soluciones entregadas argumente qué tipo de procesos se generan y consígnelo en esta sección del informe.

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 los programas entregados, 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, `mcdTFA`, `mcdEB`, `fibonacciA`, y `fibonacciI`, 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 de corrección`, debe ser entregado vía el campus virtual, a más tardar a las 10 a.m. del jueves 13 de octubre de 2022, 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.