Programación Funcional y Concurrente

Alejandra Osorio Giraldo - 2266128
Luis Manuel Cardona Trochez - 2059942

Carlos Andres Delgado Saavedra

Universidad del Valle
Sede Tuluá
2024

INFORME DE PROCESOS

Comenzaremos el informe de procesos con el primer punto

1) **Número Máximo en una lista de enteros**

Recursión Lineal

```
def maxLin(lst: List[Int]): Int = {
  if(lst.isEmpty) 0
  else if(lst.head > maxLin(lst.tail)) lst.head
  else maxLin(lst.tail)
}
```

Usaremos el ejemplo con la lista

[3,1,6,9,2]

```
println("Este caso da como resultado: " + caso1.maxLin(List(3,1,6,2,9)))//Debe imprimir 9
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
Este caso da como resultado: 9
```

Recursión de cola

```
def maxIt(lst: List[Int]): Int = {
  @tailrec
  def maxItAux(lst: List[Int], max: Int): Int = {
      if (lst.isEmpty) max
      else if (lst.head > max) maxItAux(lst.tail, lst.head)
      else maxItAux(lst.tail, max)
  }
  // Iniciamos la recursión con el primer elemento de la lista
  maxItAux(lst.tail, lst.head)
}
```

Usaremos el ejemplo con la lista

[3,1,6,66,2,9,20]

```
println("Este caso da como resultado: " + caso1.maxIt(List(3,1,6,66,2,9,20)))//Debe imprimir 66
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

> Task :app:run
Este caso da como resultado: 66
```

2) **Torres Hanoi**

```
def movsTorresHanoi(n: Int): BigInt = {
  if (n == 1) 1;
  else 2 * movsTorresHanoi(n - 1) + 1;
}
```

Dando el valor de n = 4 y n=3 tenemos que:

```
println("Los movimientos necesarios son: " + caso2.movsTorresHanoi(4))//Debe imprimir 15
println("Los movimientos necesarios son: " + caso2.movsTorresHanoi(3))//Debe imprimir 7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
> Task :app:run
Los movimientos necesarios son: 15
Los movimientos necesarios son: 7
```

Movimiento de discos

```
def torresHanoi(n: Int, t1: Int, t2: Int, t3: Int): List[(Int, Int)] = {
  if (n == 1) List((t1, t3));
  else {
    torresHanoi(n - 1, t1, t3, t2) ::: List((t1, t3)) ::: torresHanoi(n - 1, t2, t1, t3)
  }
}
```

Si hacemos la prueba con n = 3, los movimientos por hacer son:

```
println("Los movimientos son: " + caso2.torresHanoi(3,1,2,3))//Debe imprimir List((1,3), (1,2), (3,2), (1,3), (2,1), (2,3), (1,3))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

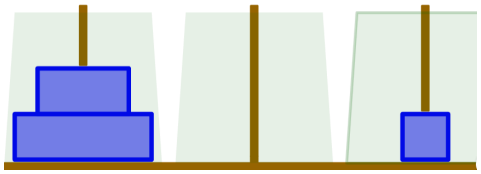
```
* Executing task: gradle: run

> Task :app:run
Los movimientos son: List((1,3), (1,2), (3,2), (1,3), (2,1), (2,3), (1,3))
```

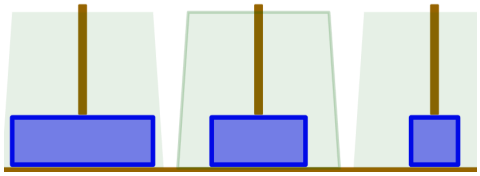
Ejemplo

Los movimientos van de la siguiente manera:

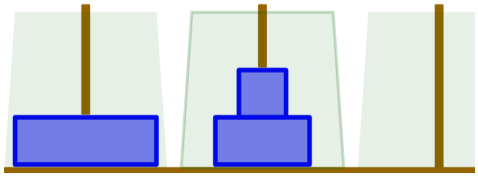
1) (1,3)



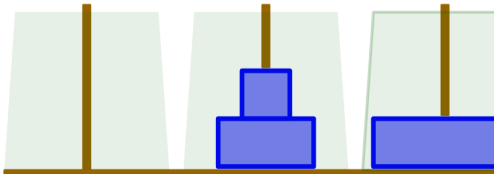
2) (1,2)



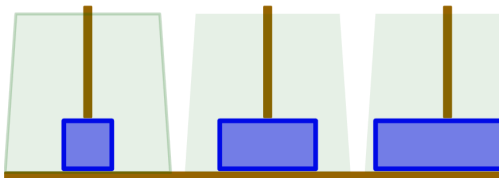
3) (3,2)



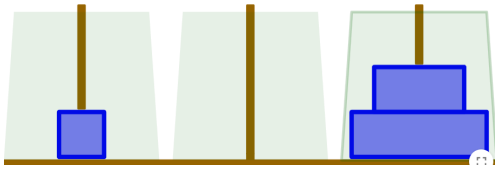
4) (1,3)



5) (2,1)



6) (2,3)



Perfect!

7) (1,3)

INFORME DE CORRECCIÓN

MaxLin

```
def maxLin(lst: List[Int]): Int = {
  if(lst.isEmpty) 0
  else if(lst.head > maxLin(lst.tail)) lst.head
  else maxLin(lst.tail)
}
```

Nuestra función recibe una lista de enteros y devuelve el mayor entero
Para ello se apoya de una condición en la cual si la lista es vacía devuelve 0
De lo contrario recibirá el primer argumento de la lista y lo comparará de manera recursiva con la misma función pero con el resto de argumentos (cola), si esto se cumple devuelve este primer elemento de la lista, en caso de que no se cumpla, retorna la cola de la función

MaxIt

```
def maxIt(lst: List[Int]): Int = {
  @tailrec
  def maxItAux(lst: List[Int], max: Int): Int = {
      if (lst.isEmpty) max
      else if (lst.head > max) maxItAux(lst.tail, lst.head)
      else maxItAux(lst.tail, max)
  }
  // Iniciamos la recursión con el primer elemento de la lista
  maxItAux(lst.tail, lst.head)
  }
```

La función **maxItAux** recibe dos parámetros, la lista y el valor máximo, este valor máximo se va actualizando en cada llamada recursiva. La ventaja de esta función es que no se desborda la pila de llamadas.
La función **maxItAux** es la función auxiliar que se encarga de hacer la recursión de cola.
Se acompaña de una condición base que es cuando la lista está vacía, en ese caso se retorna el valor máximo, de lo contrario se compara el primer elemento de la lista con el valor máximo, si el primer elemento es mayor que el valor máximo, se llama a la función **maxItAux** con la cola de la lista y el primer elemento de la lista como nuevo valor máximo, de lo contrario se llama a la función **maxItAux** con la cola de la lista y el valor máximo.

movsTorresHanoi

```
  def movsTorresHanoi(n: Int): BigInt = {
    if (n == 1) 1;
    else 2 * movsTorresHanoi(n - 1) + 1;
}
```

La función movsTorresHanoi es una función recursiva que calcula el número de movimientos necesarios para resolver el problema de las torres de Hanoi con n discos. La función se define de la siguiente manera:

- Si n es igual a 1, entonces la función retorna 1.
- En otro caso, la función retorna 2 multiplicado por el resultado de la función movsTorresHanoi con n-1 discos más 1.

torresHanoi

```
def torresHanoi(n: Int, t1: Int, t2: Int, t3: Int): List[(Int, Int)] = {
  if (n == 1) List((t1, t3));
  else {
    torresHanoi(n - 1, t1, t3, t2) ::: List((t1, t3)) ::: torresHanoi(n - 1, t2, t1, t3)
  }
}
```

La función se define de la siguiente manera:

- Si n es igual a 1, entonces la función retorna una lista con un solo elemento que representa el movimiento de un disco de la torre t1 a la torre t3.

En otro caso, la función retorna la concatenación de los siguientes elementos:

- La lista de movimientos necesarios para resolver el problema de las torres de Hanoi con n-1 discos moviendo los discos de la torre t1 a la torre t2.
- Un solo movimiento que representa el movimiento de un disco de la torre t1 a la torre t3.
- La lista de movimientos necesarios para resolver el problema de las torres de Hanoi con n-1 discos moviendo los discos de la torre t2 a la torre t3.

CASOS DE PRUEBA

MaxLin

```
9     //MaxLin
10    println("Valor de la prueba: "+ caso1.maxLin(List(1, 2, 3, 4, 5, 70,22,110,6, 7, 8, 9, 10)))
11    println("Valor de la prueba: "+ caso1.maxLin(List(3,1,6,2,9)))
12    println("Valor de la prueba: "+ caso1.maxLin(List(3,1,6,66,2,9,20)))
13    println("Valor de la prueba: "+ caso1.maxLin(List(3,7,2,5)))
14    println("Valor de la prueba: "+ caso1.maxLin(List(3,1,6,2,9,13,14,21,1)))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

> Task :app:run
Valor de la prueba: 110
Valor de la prueba: 9
Valor de la prueba: 66
Valor de la prueba: 7
Valor de la prueba: 21
```

MaxIt

```
16    //MaxIt
17    println("Valor de la prueba: "+ caso1.maxIt(List(1,8,9,2,3,4)))
18    println("Valor de la prueba: "+ caso1.maxIt(List(3,1,6,14)))
19    println("Valor de la prueba: "+ caso1.maxIt(List(3,1,6,66,2,9,20)))
20    println("Valor de la prueba: "+ caso1.maxIt(List(3,7,2,5)))
21    println("Valor de la prueba: "+ caso1.maxIt(List(3,1,6,2,9,13,14,21,1)))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

> Task :app:run
Valor de la prueba: 9
Valor de la prueba: 14
Valor de la prueba: 66
Valor de la prueba: 7
Valor de la prueba: 21
```

TorresHanoi

Cantidad de movimientos

```
25  val caso3 = new TorresHanoi()
26  println("Se requieren " + caso3.movsTorresHanoi(1) + " movimientos")
27  println("Se requieren " + caso3.movsTorresHanoi(2) + " movimientos")
28  println("Se requieren " + caso3.movsTorresHanoi(3) + " movimientos")
29  println("Se requieren " + caso3.movsTorresHanoi(4) + " movimientos")
30  println("Se requieren " + caso3.movsTorresHanoi(5) + " movimientos")
31
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
> Task :app:run
Se requieren 1 movimientos
Se requieren 3 movimientos
Se requieren 7 movimientos
Se requieren 15 movimientos
Se requieren 31 movimientos
```

Movimiento de discos entre torres

```
24  //pruebas para hanoi
25  val caso3 = new TorresHanoi()
26  println(caso3.torresHanoi(1,1,2,3))
27  println(caso3.torresHanoi(2,1,2,3))
28  println(caso3.torresHanoi(3,1,2,3))
29  println(caso3.torresHanoi(4,1,2,3))
30  println(caso3.torresHanoi(5,1,2,3))
31
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
> Task :app:run
List((1,3))
List((1,2), (1,3), (2,3))
List((1,3), (1,2), (3,2), (1,3), (2,1), (2,3), (1,3))
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))
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))

BUILD SUCCESSFUL in 4s
2 actionable tasks: 2 executed
```