Computación Distribuids
Práctica 1

Pablo Gerardo González López
pablog@ciencias.unam.mx

Daniela Susana Vega Monroy
danisu@ciencias.unam.mx

Luis Fernando Yang Fong Baeza
fernandofong@ciencias.unam.mx

Diego Estrada Mejia
diegoe@ciencias.unam.mx

Miguel Ángel Piña Avelino
miguel_pinia@ciencias.unam.mx

5 de octubre de 2021

1. Objetivo de la práctica

El objetivo de esta práctica es tener una buena introducción a Elixir como lenguaje de programación distribuido, al final de esta práctica, los alumnos deberían de ser capaces de manejar los siguientes conceptos en Elixir.

1. Operaciones básicas, llamadas a función, funcionamiento de la consola, conceptos básicos, cómo saber qué regresa una función y notación $|>$.
2. Creación de módulos y entendimientos de los mismos, imports, alias, caza de patrones y `struct`.
3. Abstracciones de estructuras, uso de MapSets, Sets, Tuples y Listas, funciones con el módulo Enum, el uso de `Enum.at`, `Enum.each`, `Enum.map`, obviamente también es el mismo caso con `MapSet.get/put` y `Map.get/put/keys`
4. La creación de procesos en BEAM con la función `spawn`, paso de mensajes y ejecuciones multiproceso bajo estados y el uso de `After`.

2. Evaluación

Cada script equivale lo mismo, 25 % de la calificación, para verificar que sus scripts sean correctos, basta con que ejecuten de manera independiente las pruebas unitarias con el comando.

3. Script de la práctica

Esta práctica es de introducción a Elixir, se deben de implementar las funciones en cada módulo declarado, así como la creación del módulo 2, con las funciones que se describen en este PDF.

El primer módulo consta de funciones básicas, el cálculo de Fibonacci y Factorial de un número n recibido como parámetro de la función, la idea es que sea una función recursiva para adaptarse al pensamiento de un lenguaje recursivo, para este módulo cada función vale exactamente lo mismo, es decir los 2,5pts de este módulo, se divide $\frac{1}{3}$ para cada función.

El módulo 2 debe de ser creado por el alumno, en él debe de contener una función llamada `test/0`, el cual cree una función lambda y regrese un `:ok` En este mismo módulo, crear otra función llamada `solve(a, b, n)` que dados tres números recibidos como parámetros, a, b y n respectivamente, determinar si $ax \equiv b \mod n$ tiene solución, como este curso tampoco se trata Álgebra Superior II o algo parecido, una congurencia de esa forma va a tener solución si y solo si, a es primo relativo con n, ejemplo:

$$5x \equiv 3 \mod 10$$

La ecuación anterior no tiene solución puesto que 5 y 10 no son primos relativos, entonces si se ejecuta la función `solve(5, 3, 10)` esta debe de regresar un `:error`.

$$5x \equiv 3 \mod 9$$

Esta ecuación sí tiene solución si se multiplica ambos lados por 2, se obtiene:

$$\cancel{2(5)}^{1}x \equiv 3(2) \mod 9$$

$$x \equiv 6 \mod 9$$

El chiste de este ejercicio es programar el cálculo del inverso multiplicativo.

La función `test/0`, equivale a 0,5pts, mientras que esta función `solve/3`, vale 2 pts.

Respecto al módulo 3, se deberá de implementar 2 funciones, eliminar los elementos duplicados de una lista recibida como parámetro, esta función se llama `elim_dup/1` y se debe de quedar con la primer ocurrencia de los elementos, la segunda función consta de implementar criba de Eratóstenes para obtener los números primos hasta n recibido como parámetro.

Ejemplos de la función `elim_dup/1`:

1. `elim_dup([1, 2, 3]` = [1, 2, 3]
2. `elim_dup([1, 1, 2, 3, 3]` = [1, 2, 3]
3. `elim_dup([1, 2, 3, 1, 2, 3]` = [1, 2, 3]
4. `elim_dup([1, 2, 3, 2, 2, 1]` = [1, 2, 3]

En cuanto a la criba, se debe de regresar una lista de elementos que contengan los primos desde 1 hasta n, el objetivo es utilizar todas las estructuras que se puedan, teniendo en mente que se pueden utilizar las funciones de la biblioteca `Enum` y la modificación de variables en Elixir.

En este caso, la primer función equivale a 1 pt y la criba de eratóstenes equivale a 1,5pts.

Por último, en el último módulo se debe de implementar un proceso que maneje las 4 estructuras vistas en la presentación del laboratorio, Listas, Tuplas, Map y MapSet, con las operaciones escritas a continuación.

Se debe de implementar utilizando paso de mensajes aunque el modelado es completamente libre y no necesariamente se le tiene que regresar algo al hilo principal excepto en las consultas, es decir, cuando se le hace un `get()` a cualquiera de las 4 estructuras.

1. Para las listas: Se debe de poder eliminar cualquier elemento de una lista, en caso de que no esté, esta operación no la modifica, agregar un elemento al final de la lista, preguntar por el tamaño de la lista.
2. Para las tuplas: Obtener la tupla completa, reemplazar el iésimo elemento y transformar la tupla a una lista.

3. Para los MapSets: Saber si un elemento x está en el conjunto, regresando `true` o `false`, obtener el número de elementos almacenados en el conjunto, sin utilizar la función `Map.size/1`, agregar un elemento al conjunto.

4. Para los Maps: Dada una llave agregar un elemento, dada una llave regresar lo que contenga y dada una llave, una lambda de cardinalidad 2 y que reciba otro operando, entonces almacene en la llave, la evaluación de la lambda con lo que está almacenado en el diccionario y el operando recibido en el mensaje.

Se recomienda aquí, pensar primero en cómo organizar los mensajes, es decir, un formato para cada mensaje y operación, para observar qué se necesita, pensar en un estado inicial para el proceso que organiza todas estas estructuras.

En este caso, se va a considerar todo el módulo como uno solo, puesto que la importancia de este es máxima, entonces si algo falla en este módulo, se considerará como malo, para obtener el punto completo, se debe de tener los cuatro puntos correctos.

4. Observaciones

No importa la complejidad en tiempo de las soluciones, salvo para la criba de eratóstenes, la única restricción es que no es válido verificar que cada número desde 1 hasta n, checar si es primo o no y regresarlo, eso no es válido.

La fecha de entrega es para el 19 de Octubre de 2021 a las 23:59:59.