COMPUTACIÓN CONCURRENTE

PRÁCTICA 6

Prof. Manuel Alcántara Juárez
manuelalcantara52@ciencias.unam.mx

Karla Vargas Godoy
karla.vargas@ciencias.unam.mx

Omar Arroyo Munguía
omar.am@ciencias.unam.mx

Ricchy Alaín Pérez Chevanier
alain.chevanier@ciencias.unam.mx

Fecha de Entrega: 29 de Mayo de 2019 a las 23:59:59pm.

1. Introducción

En esta práctica ejercitarás un los conceptos de *thread pools* y *completable futures* en Java[1] . Puedes utilizar directamenete de la documentación del lenguaje para un entendimiento más profundo de estas características del lenguaje. O recomendamos alguno de los siguientes enlaces:

1. https://www.baeldung.com/java-completablefuture
2. https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
3. https://www.callicoder.com/java-executor-service-and-thread-pool-tutorial/

En esta práctica trabajarás con una base de código construida con Java 8 y Maven 3.

En el código que recibirás junto con este documento, prodrás encontrar la clase `App` que tiene un método *main* que puedes ejecutar como cualquier programa escrito en *Java*. Para eso primero tienes que empaquetar la aplicación y después ejecutar el *jar* generado. Utiliza comandos como los siguientes:

[1]Los mismo conceptos existen en otros lenguajes, por ejemplo en Javascript las promesas son esencialmente iguales a un completable future en Java

```
$ mvn package
...
...
$ java -jar target/practica06-1.0.jar
```

Recuerda que para ejecutar las pruebas unitarias de la misma es necesario ejecutar el siguiente comando:

```
$ mvn test
```

2. Ejercicios

2.1. Merge Sort

Escribe el algoritmo de ***merge sort*** de forma paralela, de tal forma que cada llamada recursiva pueda ser atendida por un thread distinto. la idea es que utilices un `Executor` para realizar la siguiente llamada recursiva.

2.2. Suma y Multiplicación de Polinomios

Sean $P(x) = \sum_{i=0}^{d} p_i x^i$ y $Q(x) = \sum_{i=0}^{d} q_i x^i$ polinomios de grado d, donde d es una potencia de 2. Podemos expresar:

$$P(x) = P_0(x) + (P_1(x)x^{d/2}),\ Q(x) = Q_0(x) + (Q_1(x)x^{d/2})$$

Donde $P_0(x)$, $P_01(x)$, $Q_0(x)$ y $Q_1(x)$ son polinomios de grado $d/2$.

La clase `Polynomial` provee métodos *put* y *get* para acceder a los coficientes, y provee un método *split* que en tiempo constante divide el polinomio $P(x)$ in dos polinomios de grado $d/2$ como explicamos anteriormente, donde los cambios en los sub polinomios generados se reflejan en el polinomio original y vice versa.

Tu tarea is escribir algoritmos paralelos para las operaciones de suma y multiplicación para esta clase de polinomios.

La suma puede descomponerse de la sigueinte manera $P(x) + Q(x) = (P_0(x) + Q_0(x)) + (Q_1(x) + P_1(x))x^{d/2}$. El producto puede descomponerse como sigue $P(x) * Q(x) = (P_0(x) * Q_0(x)) + (Q_0(x) * P_1(x) + Q_1(x) * P_0(x))x^{d/2} + (Q_1(x) * P_1(x))$.

Para implentar este algoritmo paralelo, ademas de utilizar un *thread pool* por medio de un `Executor`, cada sub polinomio debe de ser regresado por in `CompletableFuture`.

En el capítulo 16 del libro de *The art of multiprocessor programming* puedes encontrar un ejemplo similar a este en el que suman y multiplican matrices de tamaño nxn.