# Computación Distribuida

Gordillo, Johann
418046090

Fernández, Alex
314338097

1 de Diciembre del 2020

## Instrucciones de ejecución

Para ejecutar las pruebas, hay que correr:

```
pytest -q Test.py
```

Es importante mencionar que en la implementación de DFS usamos conjuntos en lugar de listas para los vecinos y para los hijos.

## Ejercicio 1: Broadcast asíncrono con relojes lógicos

La implementación consiste en una clase *NodoBroadcast*. El mensaje que haremos llegar a todos los nodos de la gráfica desde el nodo raíz es una cadena *Hello, Graph*.

Agregamos un atributo *reloj* que será el reloj de lamport de cada procesador, en nuestra implementación es un entero incialmente cero. Cuando un proceso manda un mensaje a otro, incluye su reloj en el mensaje, y cuando un proceso recibe un mensaje, éste cambia su reloj para quedarse con el máximo entre su reloj actual y el reloj recibido. Además, un proceso incrementa su contador antes de cada evento en ese proceso.
Para simular la asincronía parcial del sistema, usamos timeouts pseudoaleatorios acotados.

En la especificación se pide guardar los eventos que vayan sucediendo. Para esto, utilizamos un atributo *eventos* representado por una lista, en la que meteremos el reloj lógico, el tipo del evento (recepción o envío), el mensaje, el *id* del proeceso que envía y el *id* del proceso que recibe el mensaje.

## Ejercicio 2: DFS asíncrono con relojes vectoriales

La implementación consiste en una clase *NodoDFS*, por convención agrego un atribtuo padre, que inicialmente será él mismo nodo. Los otros atributos son los mismos que en la práctica anterior: un identificador, un conjunto de vecinos y los canales de entrada y salida.

Para diferenciar los tipos de mensaje GO y BACK uso tuplas de tres elementos: El primer elemento es el tipo de mensaje (una cadena GO o BACK), el segundo elemento es el identificador del proceso que envía el mensaje, y el tercer elemento es el conjunto de procesos visitados.
En mi implementación, y para facilitar las operaciones dentro del algoritmo, los atributos *padre* e *hijos* son conjuntos, no listas.

La representación de los relojes vectoriales la hago con un atributo *reloj* que es una lista de $N$ entradas donde $N$ es el número de procesadores en el sistema. Inicialmente, todas las entradas son cero y cada vez que un proceso manda un mensaje, manda a su vector junto con él. Luego, cuando un proceso recibe un mensaje, incrementa su entrada en el vector en 1, y actualiza cada elemento en su vector tomando el máximo de cada entrada entre su vector y el vector que recibió. Para la actualización de los vectores, la implementación se auxilia de una función *actualiza_reloj* específica para esta tarea.

## Referencias

[1] Raynal, M. *Distributed Algorithms for Message-Passing Systems*. Springer, 2013.