# Tarea 1: Introducción

July 10, 2017

El objetivo de esta tarea es entender a mayor profundidad algunos de los componentes utilizados para construir blockchains.

Esta tarea intenta fomentar el trabajo de equipo y el intercambio de conocimientos entre personas de distintas facultades. Por eso mismo, por favor trabajen el deber juntos durante el periodo de clase o en cualquier otro momento de la semana. Solo se entregar una hoja por clase.

Este deber requiere un poco de programación, la cual sera hecha por los alumnos de ciencias de la computación. Sin embargo, el objetivo es que todos los estudiantes se familarizen con estos conceptos ya que seran utilizados a lo largo del curso. Por eso pido que todos trabajen juntos y tengan discusiones para que todos puedan entender los conceptos. La tarea es corta y el codigo necesario es simple para que sea facil que el grupo lleve un ritmo harmonioso.

Este deber sera entregado como un programa de python llamado *tarea1.py* el cual debe ser colocado en la carpeta "Tareas/Tarea1/"" del repositorio de github de esta clase: `https://github.com/netogallo/megaproyecto2017-2018`.

## 1 Preparación

1. Los estudiantes de ciencias de la computación deben crear una cuenta en github.com y enviarme su usuario asi los agrego al repositorio de esta clase que se encuentra en: `https://github.com/netogallo/megaproyecto2017-2018`
2. Para esta tarea se necesitara instalar Python 3.
3. Durante este curso se utilizara git para llevar control de versiones y Github para publicar avances. Los estudiantes de ciencias de la computacin, si no estan familiarizados con git, por favor seguir un tutorial.

## 2 Hashes Criptograficos

Un hash criptografico es una función $h : \{0,1\}^* \to \{0,1\}^n$ que recibe como entrada información representada en binario y produce un numero binario de

$n$ digitos como salida. La asignación de valores $v \in \{0,1\}^n$ del rango de $h$ (tambien llamados hashes) a las entradas $e \in \{0,1\}^*$ del dominio de $h$ debe suceder de tal forma que simule una asiganción aleatoria. En otras palabras, si uno solamente tiene un valor $v \in \{0,1\}$ resultante de aplicar el hash $h$ a alguna entrada desconocida $e \in \{0,1\}^*$, es dificil determinar cual fue la entrada $e$ que se utilizo.

En este incisio utilizaremos la biblioteca *hashlib* de python para familiarizarnos con los hashes cryptograficos. Ahora abra una terminal de python y ejecute el siguiente codigo.

```
import hashlib
hashlib.sha256(b"Hola mundo").hexdigest()
```

Esto imprime la representación hexadecimal del hash criptografico *sha-256* aplicado a la representacion binaria de "Hola mundo". Este es el hash utilizado por los Bitcoins.

Para esta tarea utlizaremos una version simplificada del hash *sha-256* llamado *sha-32*. Esta función aplica el *sha-256* a la entrada y toma los primerso 8 caracteres de la representació hexadecimal del resultado y retorna eso como resultado. Por ejemplo:

```
sha32(b"Hola mundo") #produce: "ca8f60b2"
```

Por favor crear una implementación de la función *sha-32* y colocarla en *tarea1.py*. Se permite utilizar la función *hashlib.sha256* de python en su implementación.

# 3 Nonce

Poco a poco iremos aprendiendo acerca de las partes que conforman un blockchain. Tal como dice el nombre, un blockchain es una cadena de blocks. Empezaremos por explorar las partes que forman un block. Como primer punto de division, vamos a dividir el block en dos partes: El *cuerpo* y el *nonce*. Para los propositos de esta hoja, el cuerpo simplemente es información que no puede modificarse, mientras que el nonce es una parte del block que puede tener un valor arbitrario. Para este deber, los blocks seran representados por el string "Block:" + nonce donde nonce es un string de 8 caracteres hexadecimales (0-9 y a-f) en minusculas. Por ejemplo: "Block:abcd1234".

Ahora proceda a calcular el valor *sha-32* para los siguientes valores:

```
print(sha32(str.encode("Block:00000000")))
print(sha32(str.encode("Block:00000001")))
print(sha32(str.encode("Block:00000002")))
```

Observe que tan solo con cambiar un caracter en la entrada, se produce un hash completamente diferente.

Para finalizar esta sección, defina una función llamada *block* en *tarea1.py* que dado un numero entero, produce el block correspondiente a ese numero. Por ejemplo:

```
block(0) #produce: "Block:00000000"
block(1) #produce: "Block:00000001"
```

# 4 Target

Los blockchains tienen asociado a ellos una dificultad. Esta dificultad indica la cantidad de recursos computacionales que se deben invertir para producir un block nuevo. Esta dificultad esta dada en función de un valor $t \in \{0,1\}^n$ llamado *target*, donde $n$ es el numero de bits generados por el hash. En el caso de *sha-32*, $n$ es 32 (8 digitos hexadecimales).

Para que un block sea aceptado en un blockchain, el hash criptografico de ese block debe ser menor al target. En nuestro ejemplo, si el target para nuestro ejemplo es "00009999" (representado hexadecimalmente), el block "Block:00000010" con hash *sha-32* de "fc6d4fc4" seria rechazado por el blockchain ya que "fc6d4fc4" $\not<$ "00009999". En cambio, el block "Block:0001bdb9" con hash "00004def" seria aceptado por el blockchain ya que "00004def" $<$ "00009999".

Como siguiente ejercicio, definir una funcion *findBlock* en *tarea1.py* que recibe como parametro un target, y retorna el primer block cuyo *sha-32* retorne un valor menor a ese target. Por ejemplo:

```
findBlock("000fffff") #produce: Block:0000023f
findBlock("0000ffff") #produce: Block:0001bdb9
findBlock("00000fff") #despues de un rato produce: Block:0033bb52
```

Como pueden observar, cada vez que se agrega un cero al target, es mucho más trabajoso encontrar un block que sea aceptado por el blockchain. En los blockchains reales, este target se cambia automaticamente en intervalos determinados (cada 1024 blocks para Bitcoin) de tal manera que a todas las computadoras que pertenecen al network de dicho blockchain les tome una cantidad fija de tiempo (10 minutos para Bitcoin) en encontrar un nonce con el cual el hash criptografico del block es menor al target.