75.04/95.12 Algoritmos y Programación II
Trabajo práctico 0: programación C++

Universidad de Buenos Aires - FIUBA
Segundo cuatrimestre de 2020

1. Objetivos

Ejercitar conceptos básicos de programación C++, implementando un programa y su correspondiente documentación que resuelva el problema descripto más abajo.

2. Alcance

Este Trabajo Práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá será entregado a través del campus virtual, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la Sección 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

Bitcoin es, posiblemente, la criptomoneda más importante de la actualidad. Los trabajos prácticos de este cuatrimestre están destinados a comprender los detalles técnicos más relevantes detrás de Bitcoin –en particular, la tecnología de **blockchain**. Para ello, trabajaremos con ALGOCHAIN, una simplificación de la blockchain orientada a capturar los conceptos esenciales de la tecnología.

En este primer acercamiento al problema, nos abocaremos a leer y procesar **transacciones** y ensamblar un **bloque** a partir de estas. Estos conceptos serán debidamente introducidos en la Sección 4.1, donde daremos una breve introducción a Bitcoin y blockchain. Una vez hecho esto, presentaremos la ALGOCHAIN en la Sección 4.2, destacando al mismo tiempo las similitudes y diferencias más importantes con la blockchain propiamente dicha. Las tareas a realizar en el presente trabajo se detallan en la Sección 4.3.

4.1. Introducción a Bitcoin y blockchain

Una *criptomoneda* es un activo digital que actúa como medio de intercambio utilizando tecnología criptográfica para asegurar la autenticidad de las transacciones. Bitcoin es, tal vez, la criptomoneda más importante en la actualidad. Propuesta en 2009 por una persona (o grupo de personas) bajo el seudónimo *Satoshi Nakamoto* [2], se caracterizó por ser la primera criptomoneda descentralizada que propuso una solución al problema de *double-spending* sin involucrar una tercera parte de confianza[1]. La idea esencial (y revolucionaria) que introdujo Bitcoin se basa en un registro descentralizado de todas las transacciones procesadas en el que cualquiera puede asentar operaciones. Este registro, replicado y distribuido en cada nodo de la red, se conoce como **blockchain**.

La blockchain no es otra cosa que una lista enlazada de *bloques*[2]. Los bloques agrupan *transacciones* y son la unidad básica de información de la blockchain (i.e., son los nodos de la lista). Cuando un usuario introduce una nueva transacción t en la red, las propiedades de la blockchain garantizan una detección eficiente de cualquier otra transacción que extraiga los fondos de la misma operación referenciada por t. En caso de que esto sucediera, se considera que el usuario está intentando hacer *double-spending* y la transacción es consecuentemente invalidada por los nodos de la red.

En lo que sigue describiremos los conceptos más importantes detrás de la blockchain. La Figura 1 provee un resumen visual de todo estos conceptos en el marco de la ALGOCHAIN.

4.1.1. Funciones de hash criptográficas

Una *función de hash criptográfica* es un algoritmo matemático que toma una cantidad arbitraria de bytes y computa una tira de bytes de una longitud fija (en adelante, un *hash*). Es importante que dichas funciones sean *one-way*, en el sentido de que sea computacionalmente inviable hacer una "ingeniería reversa" sobre la salida para reconstruir una posible entrada. Otra propiedad que suelen tener dichas funciones es un *efecto avalancha* en el que cambios incluso en bits aislados de la entrada derivan en hashes significativamente diferentes.

Bitcoin emplea la función de hash SHA256, ampliamente utilizada en una gran variedad de protocolos de autenticación y encricpión [3]. Esta genera una salida de 32 bytes que usualmente se representa mediante 64 dígitos hexadecimales. A modo de ejemplo, el valor de SHA256(`'Sarasa.'`) es

```
9c231858fa5fef160c1e7ecfa333df51e72ec04e9c550a57c59f22fe8bb10df2
```

4.1.2. Direcciones y firmas digitales

Una *dirección* de Bitcoin es básicamente un hash de 160 bits de la clave pública de un usuario. Mediante algoritmos criptográficos asimétricos, los usuarios pueden generar pares de claves mutuamente asociadas (pública y privada). La *clave privada* se emplea para *firmar*

[1] El doble gasto (o *double-spending*) ocurre cuando el emisor del dinero crea más de una transacción a partir de una misma operación previa. Naturalmente, sólo una de las nuevas transacciones debería ser válida puesto que, de lo contrario, el emisor estaría multiplicando dinero.

[2] Técnicamente, la blockchain es más bien un árbol de bloques, pero esto será abordado en el contexto del siguiente trabajo práctico.

los mensajes que desean transmitirse. Cualquier receptor puede luego verificar que la firma es válida utilizando la *clave pública* asociada. Esta clase de métodos criptográficos ofrecen garantías de que es computacionalmente difícil reconstruir la clave privada a partir de la información públicamente disponible.

4.1.3. Transacciones

Una *transacción* en Bitcoin está definida por una lista de entradas (*inputs*) y otra de salidas (*outputs*). Un *output* se representa a través de un par (`value`, `addr`), donde `value` indica la cantidad de bitcoins que recibirá el destinatario y `addr` es el hash criptográfico de la clave pública del destinatario. Por otro lado, un *input* puede entenderse como una tupla (`tx_id`, `idx`, `key`, `sig`) tal que:

- `tx_id` indica el hash de una transacción previa de la que esta nueva transacción toma fondos,
- `idx` es un índice dentro de la secuencia de *outputs* de dicha transacción (los fondos de este *input*, luego, provienen de dicho *output*),
- `key` es la clave pública asociada a tal *output*, y
- `sig` es la firma digital del hash de la transacción usando la clave privada asociada a la clave pública del *output*.

En consecuencia, cada *input* hace referencia a un *output* anterior en la blockchain (los campos `tx_id` y `idx` suelen agruparse en una estructura común bajo el nombre de *outpoint*). Para verificar que el uso de dicho *output* es legítimo, se calcula el hash de la clave pública y se verifica que sea igual a la que figura en el *output* utilizado. Luego, basta con verificar la firma digital con esa clave pública para asegurar la autenticidad de la operación.

Para garantizar la validez de una transacción, es importante verificar no sólo que cada *input* es válido sino también que la suma de los *outputs* referenciados sea mayor o igual que la suma de los *outputs* de la transacción. La diferencia entre ambas sumas, en caso de existir, es lo que se conoce como *transaction fee*. Este valor puede ser reclamado por quien agrega la transacción a la blockchain (como retribución por suministrar poder de cómputo para realizar el minado de un nuevo bloque).

Naturalmente, una vez que un *output* de una transacción haya sido utilizado, este no podrá volver a utilizarse en el futuro. En otras palabras, cada nueva transacción sólo puede referenciar *outputs* que no fueron utilizados previamente. Estos últimos se conocen como *unspent transaction outputs* (UTXOs).

4.1.4. Bloques

Toda transacción de Bitcoin pertenece necesariamente a un *bloque*. Cada bloque está integrado por un encabezado (*header*) y un cuerpo (*body*). En el header se destaca la siguiente información:

- El hash del bloque antecesor en la blockchain (`prev_block`),

- El hash de todas las transacciones incluidas en el bloque (`txns_hash`),
- La *dificultad* con la cual este bloque fue ensamblado (`bits`), y
- Un campo en el que se puede poner datos arbitrarios, permitiendo así alterar el hash resultante (`nonce`).

El cuerpo de un bloque, por otro lado, incluye la cantidad total de transacciones (`txn_count`) seguido de la secuencia conformada por dichas transacciones (`txns`).

4.1.5. Minado de bloques

Para que un bloque sea válido y pueda en consecuencia ser aceptado por la red de Bitcoin, debe contar con una prueba de trabajo (*proof-of-work*) que debe ser difícil de calcular y, en simultáneo, fácil de verificar. El mecanismo detrás de este proceso se conoce como *minado*. Las entidades encargadas de agrupar transacciones y ensamblar bloques válidos son los *mineros*.

Los mineros obtienen una recompensa en bitcoins cuando agregan un bloque a la blockchain. Esto último se logra calculando la *proof-of-work* del nuevo bloque, lo cual a su vez se realiza con poder de cómputo. La *proof-of-work* de un bloque consiste en un hash $h = \text{SHA256}(\text{SHA256}(\text{header}))$ tal que su cantidad de ceros en los bits más significativos es mayor o igual que un valor derivado del campo `bits` del header del bloque. A los efectos prácticos, consideraremos que, si el campo `bits` indica un valor d, la cantidad de ceros en los bits más significativos de h debe ser $\geq d$.

El esfuerzo necesario para ensamblar un bloque que cumpla con la dificultad de la red crece exponencialmente con la cantidad de ceros requerida. Esto se debe a que agregar un cero extra a la dificultad disminuye a la mitad la cantidad de hashes que cumplan con dicha restricción. No obstante esto, para verificar que un bloque cumple con esta propiedad, basta con computar dos veces la función de hash SHA256. De esta forma, se puede comprobar fácilmente que un minero realizó una cierta cantidad de trabajo para hallar un bloque válido.

4.2. Algochain: la blockchain de Algoritmos II

La blockchain simplificada con la que estaremos trabajando a lo largo del cuatrimestre es la Algochain. Al igual que la blockchain, la Algochain se compone de bloques que agrupan transacciones. A su vez, las transacciones constan de una secuencia de *inputs* y otra de *outputs* que siguen los mismos lineamientos esbozados más arriba. La Figura 1 muestra un esquema de alto nivel de la Algochain.

4.2.1. Direcciones

Una de las diferencias más importantes con la blockchain radica en una simplificación intencional del proceso de verificación y validación de direcciones al momento de procesar las transacciones: la Algochain no utiliza firmas digitales ni claves públicas. En su lugar, tanto los *inputs* como los *outputs* de las transacciones referencian directamente direcciones de origen y destino de los fondos, respectivamente. Esta *dirección* la interpretaremos como un hash SHA256 de una cadena de caracteres arbitraria que simbolice la dirección propiamente dicha

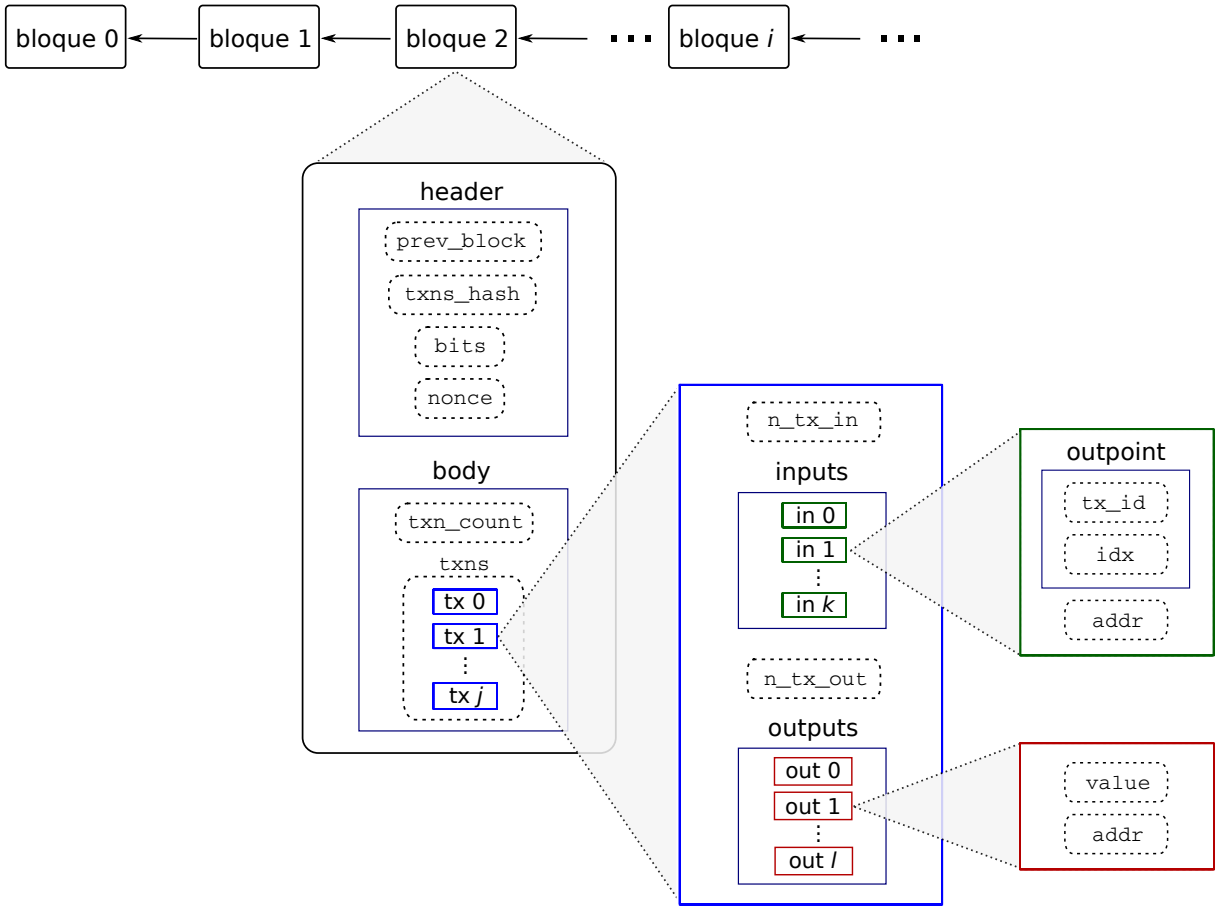

Figura 1: Esquema de alto nivel de la ALGOCHAIN

del usuario. A la hora de procesar una nueva transacción t, simplemente deberá validarse que la dirección `addr` de cada *input* de t coincida exactamente con el valor del `addr` especificado en el *output* referenciado en dicho *input*.

A modo de ejemplo, si la dirección real de un usuario de nuestra ALGOCHAIN fuese `Segurola y Habana`, los campos `addr` de las transacciones que involucren a dicho usuario deberían contener el valor `addr` = SHA256(`'Segurola y Habana'`), que equivale a

```
485c8c85be20ebb6a9f6dd586b0f9eb6163aa0db1c6e29185b3c6cd1f7b15e9e
```

4.2.2. Hashes de bloques y transacciones

Tal como ocurre en blockchain, y como explicamos en la Sección 4.1, en ALGOCHAIN identificamos unívocamente bloques y transacciones mediante hashes SHA256 dobles.

El campo `prev_block` del header de un bloque b indica el hash del bloque antecesor b' en la ALGOCHAIN. De este modo, `prev_block` = SHA256(SHA256(b')). Dicho hash lo calcularemos sobre una concatenación secuencial de todos los campos de b' respetando exactamente el formato de bloque que describiremos en la Sección 4.4.

De forma análoga, el campo `tx_id` de los *inputs* de las transacciones lo calcularemos con un doble hash SHA256 sobre una concatenación de todos los campos de la transacción correspondiente.

Finalmente, el campo `txns_hash` del header de un bloque b contendrá también un doble hash SHA256 de todas las transacciones incluidas en b. En el contexto de este trabajo práctico, dicho hash lo calcularemos sobre una concatentación de todas las transacciones respetando exactamente el formato que describiremos en la Sección 4.4. En otras palabras, dadas las transacciones $t_0, t_1 \ldots, t_j$ del bloque b,

$$\texttt{txns\_hash} = \text{SHA256}(\text{SHA256}(t_0\, t_1\, \ldots\, t_j))$$

4.3. Tareas a realizar

Para apuntalar los objetivos esenciales de este trabajo práctico, esbozados en la Sección 1, deberemos escribir un programa que reciba transacciones por un *stream* de entrada y ensamble un bloque con todas ellas una vez finalizada la lectura. Dicho bloque deberá escribirse en un *stream* de salida. De este modo, al estar trabajando con único bloque, por convención dejaremos fijo el valor del campo `prev_block` en su header. Dicho campo debe instanciarse en

```
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

Naturalmente, nuestros bloques deben satisfacer los requisitos de validez delineados en la Sección 4.1.5. En particular, nos interesa exhibir la correspondiente *proof-of-work* para poder reclamar las eventuales recompensas derivadas del minado. Para ello, nuestros programas recibirán como parámetro el valor de la *dificultad* d esperada. En otras palabras, debemos garantizar que la cantidad de ceros en los bits más significativos de nuestro hash h es $\geq d$, siendo $h = \text{SHA256}(\text{SHA256}(\text{header}))$. Recordar que, en caso de no encontrar un hash h válido, es posible intentar sucesivas veces modificando el campo `nonce` del header del bloque (la Sección 4.4 describe en detalle el formato de dicho header). Este campo puede instanciarse con valores numéricos arbitrarios tantas veces como sea necesario hasta dar con un hash válido.

Para simplificar el proceso de desarrollo, la cátedra suministrará código C++ para calcular hashes SHA256.

4.4. Formatos de la Algochain

En esta Sección detallaremos el formato de las transacciones y bloques de la ALGOCHAIN. Tener en cuenta que es sumamente importante **respetar de manera estricta** este formato. Mostraremos algunos ejemplos concretos en la Sección 4.6.

4.4.1. Transacciones

Toda transacción de la ALGOCHAIN debe satisfacer el siguiente formato:

- Empieza con una línea que contiene el campo entero `n_tx_in`, que indica la cantidad total de *inputs*.

- Luego siguen los *inputs*, uno por línea. Cada *input* consta de tres campos separados entre sí por un único espacio:
  - `tx_id`, el hash de la transacción de donde este *input* toma fondos,
  - `idx`, un valor entero no negativo que sirve de índice sobre la secuencia de *outputs* de la transacción con hash `tx_id`, y
  - `addr`, la dirección de origen de los fondos (que debe coincidir con la dirección del *output* referenciado).
- Luego de la secuencia de *inputs*, sigue una línea con el campo entero `n_tx_out`, que indica la cantidad total de *outputs* en la transacción.
- Las `n_tx_out` líneas siguientes contienen la secuencia de *outputs*, uno por línea. Cada *output* consta de los siguientes campos, separados por un único espacio:
  - `value`, un número de punto flotante que representa la cantidad de Algocoins a transferir en este *output*, y
  - `addr`, la dirección de destino de tales fondos.

4.4.2. Bloques

Como indicamos en la Sección 4.1.4, todo bloque arranca con un header. El formato de nuestros headers es el siguiente:

- El primer campo es `prev_block`, que contiene el hash del bloque completo que antecede al bloque actual en la ALGOCHAIN.
- Luego sigue el campo `txns_hash`, que contiene el hash de todas las transacciones incluidas en el bloque. El cálculo de este hash debe realizarse de acuerdo a las instrucciones de la Sección 4.2.2.
- A continuación sigue el campo `bits`, un valor entero positivo que indica la dificultad con la que fue minada este bloque.
- El último campo del header es el `nonce`, un valor entero no negativo que puede contener valores arbitrarios. El objetivo de este campo es tener un espacio de prueba modificable para poder generar hashes sucesivos hasta satisfacer la dificultad del minado.

Todos estos campos deben aprecer en líneas independientes. En la línea inmediatamente posterior al `nonce` comienza la información del body del bloque:

- La primera línea contiene el campo `txn_count`, un valor entero positivo que indica la cantidad total de transacciones incluidas en el bloque.
- A continuación siguen una por una las `txn_count` transacciones. Todas ellas deben respetar el formato de transacción de la Sección anterior.

4.5. Interfaz

La interacción con nuestros programas se dará a través de la línea de comando. Las opciones a implementar en este trabajo práctico son las siguientes:

- `-d`, o `--difficulty`, que indica la dificultad esperada d del minado del bloque. En otras palabras, el hash $h = \text{SHA256}(\text{SHA256}(\text{header}))$ debe ser tal que la cantidad de ceros en sus bits más signficativos sea $\geq d$. Esta opción es de carácter obligatorio (i.e., el programa no puede continuar en su ausencia).
- `-i`, o `--input`, que permite controlar el stream de entrada de las transacciones. El programa deberá recibir las transacciones a partir del archivo con el nombre pasado como argumento. Si dicho argumento es "`-`", el programa las leerá de la entrada standard, `std::cin`.
- `-o`, o `--output`, que permite direccionar la salida al archivo pasado como argumento o, de manera similar a la anterior, a la salida standard -`std::cout`- si el argumento es "`-`".

4.6. Ejemplos

Consideremos la siguiente transacción t:

```
1
48df0779 2 d4cc51bb
1
250.5 842f33e9
```

Por una cuestión de espacio, los hashes involucrados en estos ejemplos aparecen representados por sus últimos 8 bytes. De este modo, el hash `tx_id` del *input* de t y las direcciones `addr` referenciadas en el *input* y en el *output* son, respectivamente,

```
26429a356b1d25b7d57c0f9a6d5fed8a290cb42374185887dcd2874548df0779
f680e0021dcaf15d161604378236937225eeecae85cc6cda09ea85fad4cc51bb
0618013fa64ac6807bdea212bbdd08ffc628dd440fa725b92a8b534a842f33e9
```

Esta transacción consta de un único *input* y un único *output*. El *input* toma fondos de alguna supuesta transacción t' cuyo hash es `48df0779`. En particular, los fondos provienen del tercer *output* de t' (observar que `idx` es 2). La dirección de origen de los fondos es `d4cc51bb`. Por otra parte, el *output* de t deposita 250,5 Algocoins en la dirección `842f33e9`.

Supongamos ahora que contamos con un archivo de transacciones que contiene la información de t:

```
$ cat txns.txt
1
48df0779 2 d4cc51bb
1
250.5 842f33e9
```

La siguiente invocación solicita ensamblar un nuevo bloque con esta transacción:

```
$ ./tp0 -i txns.txt -o block.txt -d 3
```

Notar que el bloque debe escribirse al archivo `block.txt`. Además, la dificultad de minado sugerida es 3: esto nos dice que el hash del header de nuestro bloque debe comenzar con 3 o más bits nulos. Una posible salida podría ser la siguiente:

```
$ cat block.txt
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
155dc94b29dce95bb2f940cdd2d7e0bce66dca9370c3ed96d50a30b3d84f8c4c
3
12238
1
1
48df0779 2 d4cc51bb
1
250.5 842f33e9
```

Observar que el valor del `nonce` es 12238. Si bien dicho `nonce` permite encontrar un hash del header satisfactorio, esta elección por supuesto no es única. El hash del header, bajo esta elección, resulta

```
08850af2009f750a59dd91d0d5d4bafa25c6421de132336e2c5ee67bba6e352e
```

Como puede verse, el hash comienza con cuatro bits nulos.

Un ejemplo aún más simple (y curioso) consiste en invocar nuestros programas con una entrada vacía:

```
$ touch empty.txt
$ ./tp0 -i empty.txt -d 3
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
cd372fb85148700fa88095e3492d3f9f5beb43e555e5ff26d95f5a6adc36f8e6
3
59329
0
```

En primer lugar, notemos que, al no especificar un *stream* de salida, el programa dirige la escritura del bloque a la salida estándar. El bloque construido, si bien no incluye ninguna transacción, contiene información válida en su header. Notar que el campo `txns_hash` se calcula en este caso a partir del doble hash SHA256 de una cadena de caracteres vacía.

4.7. Portabilidad

Es deseable que la implementación desarrollada provea un grado mínimo de portabilidad. Sugerimos verificar nuestros programas en alguna versión reciente de UNIX: BSD o Linux.

5. Informe

El informe deberá incluir, como mínimo:

- Una carátula que incluya los nombres de los integrantes y el listado de todas las entregas realizadas hasta ese momento, con sus respectivas fechas.
- Documentación relevante al diseño e implementación del programa.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C++.
- Este enunciado.

6. Fechas

La última fecha de entrega es el **jueves 12 de noviembre de 2020**.

Referencias

[1] Wikipedia, "Bitcoin Wiki." `https://en.bitcoin.it/wiki/Main_Page`.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.

[3] Wikipedia, "SHA-2." `https://en.wikipedia.org/wiki/SHA-2`.