IIC3272 — Criptomonedas y contratos inteligentes

Tarea 3: Scroogecoin

Entrega: Viernes – 3 de Octubre – 2025 – 23:59 (hora Chile continental)

1. Cosas administrativas

Cada tarea en este curso equivale a 20 % de la nota final. En total vamos a tener 6 tareas, pero la peor tarea que solucionan no cuenta para la nota final. Quiere decir que pueden botar una tarea.

El programa con su solución hay que subir por el buzon de canvas.

Uso de materiales (o soluciones) encontradas en Internet está permitido. Solo se les pide citar la fuente que utilizaron. No se aplica ninguna penalidad para su uso.

2. La tarea

En esta tarea deben implementar el majeo de transacciones y formación de bloques por Scrooge en el sistema Scroogecoin.

El código entregado contiene varias clases y archivos, pero el único archivo relevante para esta tarea es `scroogecoin.py`, en el cual tendrán que implementar el método `process_transactions` cual permite al Scrooge armar bloques nuevos y poner las transacciones válidas a estos bloques.

Otros archivos relevantes para esta tarea son:

- `transaction.py` – cual contiene la implementación de la lógica de una transacción de Scroogecoin; esto es lo mismo lo que revisamos en clases.
- `blockchain.py` – cual contiene la lógica del blockchain de Scroogecoin.
- `keys.py` – cual contiene las llaves ocupadas en el sistema.

A cambio de lo visto en clases, ahora cada bloque publicado por Scrooge (potencialmente) contiene más de una transacción. En particular (para hacer testing fácil) asumiremos que un bloque debe contener dos transacciones si hay dos transacciones válidas por agregar y solo una transacción si hay solo una transacción válida.

Para operar Scroogecoin nuestro amigo Scrooge mantiene tres estructuras de datos:

- `transactions`: un dicionario de todas las transacciones válidas (cuales se publican en el blockchain). Aquí la llave es hash de la transacción y el valor es la transacción (cómo el objeto de la clase `Transaction` vista en clases).
- `blockchain`: un objeto de clase `Blockchain` dónde cada bloque viene firmado por Scrooge. La lógica de esta clase la explicaremos más abajo.
- `utxo_pool`: un `set()` de tuplas de forma `(txID, nrOutput)`, dónde `txID` es el hash de la transacción, y `nrOutput` en número del output de esta transacción cual todavía no fue gastado.

Reglas de Scroogecoin:

- Cada transacción debe tener firmas válidas.
- Valor de todos los inputs debe ser mayor o igual al valor de todos los outputs.
- Para gastar un input este no puede ser gastado anteriormente.
- Las transacciones se procesan en su órden de llegada.
- Cada bloque de blockhain puede contener cómo máximo dos transacciones.
- Los bloques debe ir llenados máximalmemnte, quiere decir, si hay dos transacciones válidas para poner en un bloque, este debe tener dos transacciones.
- Cada bloque es firmado por Scrooge (ya implementado).
- Si hay dos transacciones en un bloque, la segunda puede gastar un output de la primera (hay tests de esto).
- Si llegan dos transacciones que intentan gastar el mismo UTXO válido, la primera que llega lo gasta.
- Si una misma transacción `createCoins` (mismos outputs) se trata de publicar dos veces, solo se publica primera ves (se identifica por su hash).

En la tarea deben completar la implementación del método `process_transactions(tx_list)` en la clase `Scroogecoin`, cual recibe cómo su input un arreglo de objetos de la clase `Transaction` y procesa las transacciones según la lógica explicada arriba, formado bloque da a dos transacciones, agregandoles al elemento `blockchain` de la clase `Scroogecoin`. Adicionalmente, las transacciones cuales aparecen en este blockchain deben estar en el elemento `transactions` de la clase, igual cómo todos los UTXOs en el elemento `utxo_pool`.

El flujo de procesar transacciones es iterar sobre el arreglo de inputs elemento por elemento. Cuando encontramos dos transacciones válidas (firmas OK, valores OK, inputs en UTXO pool, valores y dueños de inputs correctos, txID del input válido y guardado en

`transactions`), estas dos se ponen en un bloque (detalles abajo). El proceso continua hasta procesar todas las transacciones en el arreglo `tx_list`. Idea es que si nuestro blockchain parte vació, todos los bloques, salvo (quizás) el último contienen dos transacciones. En el caso que no hay ninguna transacción válida todo debe funcionar igual.

Su implementación debe funcionar de tal manera que al final de este proceso el contenido de ***`transactions`*** *contiene solo las transacciones cuales aparecen en el blockchain,* ***`blockchain`*** *contiene los bloques firmados por Scrooge, y* ***`utxo_pool`*** *contiene todos los UTXOs todavía no gastados. El contenido de estos tres elementos debe ser correcto para que su tarea sea correcta.*

¿Cómo funciona blockchain de Scroogecoin? El blockchain de Scroogecoin viene completamente implementado en el archivo `blockchain.py`. El blockchain mismo guarda bloques indexados por sus hashes y tiene el método `add_block(block)` cual permite agregar un bloque nuevo de manera consistente. Otro elemento fundamental de la clase `Blockchain` es `head`, cual contiene el hash de la cabeza actual del blockchain. La cabeza se asigna automaticamente al agregar un bloque.

Un bloque en Scroogecoin se inicia con dos parámetros:

- `tx_hashes` – un arreglo de hashes cuales aparecen en este bloque. Estos objetos deben ser en el formato hexadecimal (ver code hints abajo).
- `prev_hash` – el hash del bloque previo. Este hash lo deben asignar al formar un bloque.

Usando esto, el bloque se completa comptando la raíz de Merkle para estos hashes, se firma por Scrooge (usando su llave privada) y se computa su hash, guardado en el campo `block_hash`.

3. Entrega

Su entrega debe ser una archivo `zip` (con el nombre que quieran) que contenga todos los archivos necesarios para que funcione su implementación. Entre estos deben estar todos los archivos publicados puesto que serán usados como dependencias para correr los tests.

4. Code hints

¿Cómo veo el hash de la transacción `tx`?

- `tx.id()`

¿Cómo agrego la transacción `tx` a `transactions`?

- `self.transactions[tx.id()] = tx`