Proyecto AVL Tree Blockchain

Estructura de Datos y Algoritmos

Grupo 3

Integrantes:

- Lóránt Mikolás
- Felipe Gorostiaga
- Tomás Ferrer
- Santiago Swinnen
- Luke Mitchell

Introducción

En este informe nos proponemos analizar y exponer las decisiones tomadas a lo largo de la realización del trabajo práctico. También presentaremos algunos de los problemas encontrados en el desarrollo y cómo fueron solucionados.

El trabajo práctico consiste en implementar una *blockchain* en la que cada uno de los nodos (o bloques) guarde operaciones sobre un árbol AVL.

Decisiones sobre el modelado

En lo que respecta al modelado del problema decidimos implementar las siguientes clases principales:

- BlockChain: la clase principal con la estructura de la blockchain y sus métodos.
- Block: inner-class de BlockChain, cada instancia de esta clase representa un bloque en la BlockChain.
- AVLTree : árbol AVL donde se realizan las operaciones (add, remove, lookup) ingresadas por comandos.
- Terminal: clase encargada del front-end, hace las verificaciones del input ingresado por el usuario y parseo del mismo.
- TreePrinter: clase encargada de imprimir el árbol AVL por pantalla

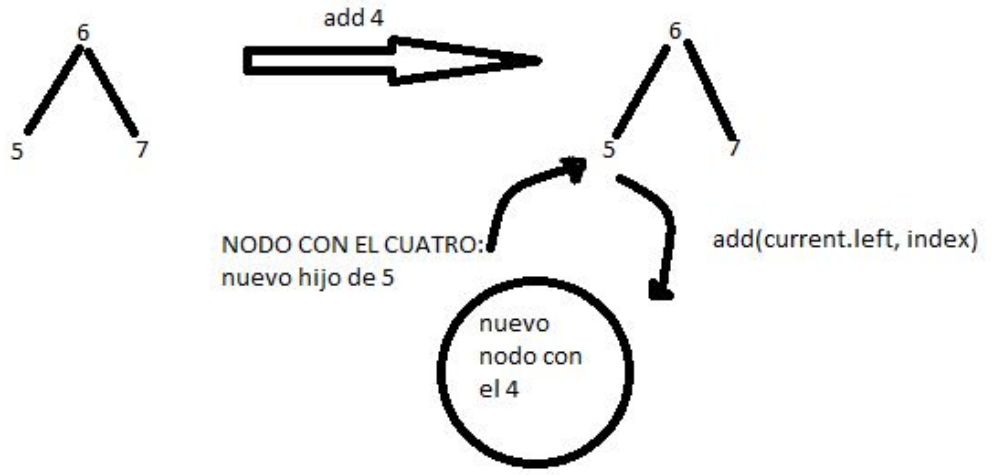
El árbol AVL

Para la construcción del árbol se utilizó una implementación similar a la de BinarySearchTree (BST) con header propuesta por la cátedra. Se le agregaron modificaciones según las especificaciones de un árbol AVL, donde la ejecución de las operaciones add() y remove() implica realizar el balance de cada nodo en el árbol, con sus respectivas rotaciones en caso de ser necesarias.

Retorno de los métodos add(), remove() y lookup().

Decidimos que era fundamental que estos métodos indiquen de alguna forma si fueron realizados con éxito o no, para luego guardar en los bloques de la blockchain si la instrucción ingresada por el usuario pudo llevarse a cabo con éxito o no (“add X true/false”).

En primer lugar, la operación add devuelve el nuevo hijo de la recursión del nodo que llamó a la recursión con su subárbol izquierdo o derecho. Esto queda ilustrado en el siguiente diagrama:



En caso de que el elemento no haya sido agregado porque ya estaba en el árbol se retorna *null.* Luego la wrapper retorna *false* al recibir null, o, de no ser así, *true*.

En segundo lugar, el *remove* retorna dos valores: uno con el nodo que será hijo del nodo que llamó a remove() con su hijo izquierdo o derecho en la recursión y otro con Boolean para indicar si la operación pudo ser realizada. Optamos por implementar una clase DataPair para poder llevar esto a cabo.

En tercer lugar, para el caso del lookup utilizamos un DataPair, donde el primer valor era un Boolean indicando el éxito o fracaso de la operación y el segundo un Set de Integers con los índices de bloques que modificaron el nodo buscado.

Uso particular del DataPair and

En el caso que el nodo a eliminar tenga dos hijos decidimos buscar el sucesor inorder para reemplazarlo. Esta operación puede resultar muy costosa si la altura del árbol es grande y el nodo a eliminar se encuentra en los primeros niveles del mismo. La implementación más popular consiste descender por el árbol para buscar la referencia al sucesor inorder y luego llamar desde el nodo a ser reemplazado a remove (nodo sucesor inorder). Esto lleva a hacer el recorrido dos veces hasta el nivel del árbol en el que se encuentra el sucesor inorder: una para la búsqueda y otra para la remoción del nodo.
Entonces decidimos hacer uso del DataPair como valor de retorno para buscar la referencia y eliminar el nodo del árbol en un única operación, en donde un elemento es el nuevo hijo del nodo que llamó a la recursión con su hijo izquierdo y el otro es la referencia al sucesor inorder.

Guardado de los índices de los bloques que modificaron el nodo

En cuanto a los índices, decidimos guardarlos en los nodos del árbol AVL en vez de en una estructura exterior. El acceso a los mismos es realizado únicamente en el comando lookup, por lo tanto ya estaríamos parados en el nodo correspondiente para hacer el retorno de los mismos.

Para almacenarlos utilizamos un HashSet de Integers para tener acceso O(1) y poder asegurarnos de que no tendríamos elementos repetidos en la colección.

Casos en los que consideramos que una operación modificó un nodo

1. Si a un nodo se le modificó uno de sus hijos.
2. Si a un nodo fue rotado a la izquierda o derecha.
3. Cuando fue agregado.

Comentario sobre TreePrinter

Vale la pena marcar que la clase TreePrinter, cuya función es dibujar el árbol en la terminal, fue extraída de la solución propuesta por un usuario en *Stack Overflow* en el siguiente *thread*:
https://stackoverflow.com/questions/4965335/how-to-print-binary-tree-diagram

Detalles en algunas operaciones

Modify

La operación modify debe ser llamada de la siguiente forma:
modify N [path]
Donde N es el índice del bloque de blockchain a modificar y path es el path absoluto al archivo con la información a insertar. Notar que el path se encierra entre corchetes.

Lookup

Queremos aclarar que lookup imprime en pantalla los índices de los bloques que modificaron al nodo buscado pero no guarda dichos índices en la blockchain. Simplemente guarda true y false como valor de retorno para indicar si la operación se realizó con éxito o no.

BlockChain

Para la implementación de la clase principal BlockChain, se usó como estructura de datos principal una ArrayList<Block> de la API de Java y una referencia al árbol AVL.
La inner-class Block fue creada para representar los bloques dentro de la lista.
Dentro de la clase Block los métodos principales implementados son calculateHash() y validHash() . El primero es el encargado de calcular el Hash para el bloque específico, para esto primero usamos una función de hash conocida (sha256) creada en 2001 por la NSA de EEUU y luego creamos nuestra propia función de hash, comparando su funcionamiento.

El segundo método es un método de instancia privado el cual recibe un hash calculado previamente como argumento y valida si el mismo es válido, es decir si contiene la cantidad de ceros indicados por el usuario con la operación “zeros X”.
Para evitar problemas y por una cuestión de comodidad la blockchain contiene un nodo inicial con referencia a un hash previo inexistente. A este bloque lo llamamos Genesis Block.

Guardado de las operaciones e indicación de éxito

Para realizar operaciones en el árbol AVL, el usuario ingresa por línea de comandos la operación indicada como se especifica en los requerimientos del TP. Terminal se encarga de validar los datos ingresados y, en caso de ser correctos, usa el método operate() de la blockchain, enviando el tipo de operación indicada y el Integer. Por medio de un switch() se realiza la operación correcta al árbol avl y dependiendo de si la operación se realizó o no correctamente se crea un nuevo bloque en la chain indicándolo en su variable de instancia “Instruction”.

Validación de la cadena

La validación de la cadena se implementó siguiendo dos especificaciones de la estructura de una blockchain. En primer lugar, el valor de la variable prevHash de un Bloque debe coincidir con el valor de la variable hash del bloque anterior, es decir:
blockchain.get(i).prevHash.equals(blockchain.get(i-1).hash) debe retornar true para todo valor de i mayor que 0. En segundo lugar, para preservar la seguridad de la blockchain y corroborar que los datos dentro de un bloque no hayan sido modificados, el hash debe permanecer con el mismo valor que se le calculó una vez que el bloque fue creado. Para esto se calcula el hash nuevamente usando el método calculateHashNoNonce(), el cual no mina un nonce que corresponda a la cantidad de ceros indicada sino que usa los datos existentes en la instancia del bloque.

Teniendo en cuenta estas dos propiedades que debe cumplir la blockchain, se recorren los bloques que la componen y se retorna ‘true’ en caso que todos la cumplan, o ‘false’ caso contrario.

Para finalizar creemos que vale la pena señalar que la cadena no se revalida una vez rota . Esta decisión fue tomada en función de las indicaciones de la cátedra.

BuildFile de ANT

El archivo se encuentra en la carpeta src del repositorio. Se debe tener en cuenta que para que el buildfile funcione se debe tener configurada correctamente la variable de entorno JAVA_HOME.

El archivo.jar se generará en la misma carpeta donde se encuentre el BuildFile de ANT.

Resultados

A continuación se muestran los resultados obtenidos para el tiempo que lleva la operatoria según la cantidad de ceros.

Cantidad de Ceros	Milisegundos (ms)
0	**0**
1	**3**
2	**20**
3	**34**
4	**352**
5	**3600**

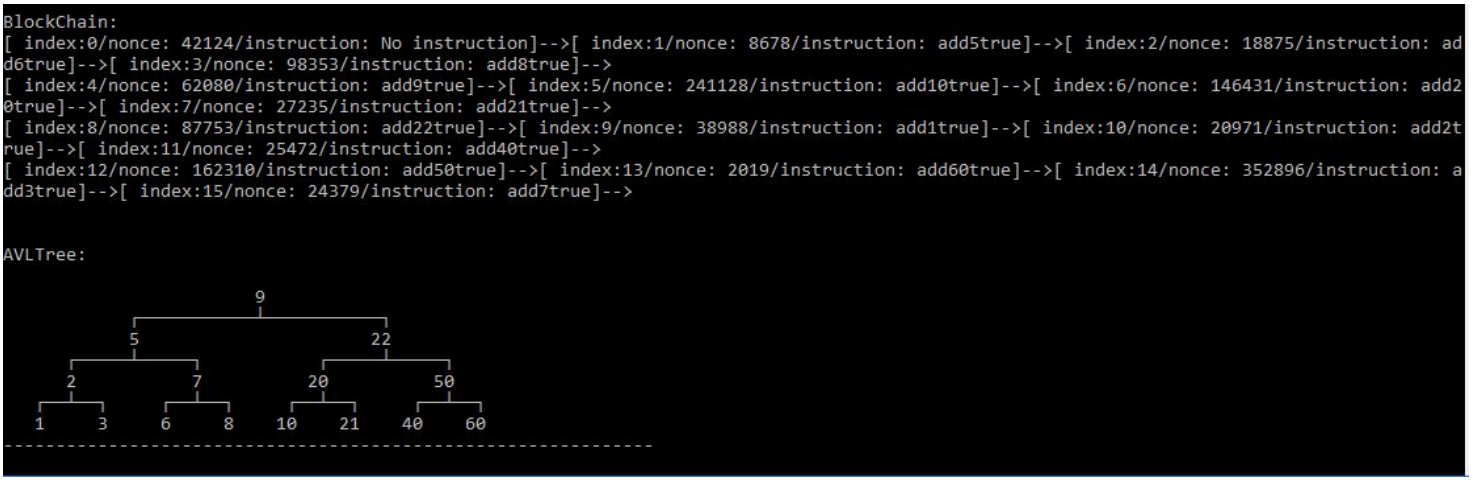
Como se puede observar en la tabla en la que se miden los tiempos de ejecución de la función sha256, la misma es muy veloz pero crece exponencialmente a la medida que se le piden más ceros en el hash. Si intentamos minar hashes con más de 5 ceros los tiempos se extienden considerablemente, a tiempos mayores a 15 minutos.

Como consecuencia de las prioridades tomadas al momento del desarrollo, de las cuales la más importantes fueron la eficiencia y la prolijidad, logramos una implementación sencilla en lo visual y rápida en la ejecución.

Al ser un programa de interacción con el usuario por consola, buscamos que ésta sea lo más clara posible. Sin embargo, se invirtió más tiempo en la algoritmia de back end que en el aspecto visual.

Imagen de un árbol de mayor tamaño

A modo ilustrativo de la interfaz, dejamos una imagen sobre cómo se ve un árbol de por lo menos 4 niveles de profundidad.



Conclusión

En conclusión, consideramos que en líneas generales se alcanzaron las metas propuestas al comienzo del trabajo. El desarrollo nos llevó a investigar y aprender mucho sobre los usos e implementaciones de *blockchains*, muy vigentes en la actualidad. Además, incorporamos conocimientos relativos a las operaciones sobre árboles, lo cual nos llevó a repensar nuestra implementación en reiteradas ocasiones. Creemos que si bien no es el objeto de este trabajo, hubiera sido interesante poder implementar una parte gráfica para la impresión del árbol y de la cadena.

Anexo

Hipervínculos de interés

- Repositorio de GitHub: https://github.com/mikolaslorant/AVLBLOCKCHAIN
- Video: https://youtu.be/ni5d8SgtCyc