Informe tp especial

Integrantes: Johnathan Katan, Joaquin Ormachea, Diego Bruno Cilla, Francisco Delgado y Federico Mamone

**Introducción:**

El objetivo de este informe es en primer lugar presentar una descripción de las estructuras de datos utilizadas para el desarrollo de la Blockchain y del árbol AVL, explicando cómo están compuestas y los métodos implementados.

En segundo lugar, se presenta una justificación de las decisiones de diseño y de por qué se decidió implementar estas estructuras de datos, comparándolas con estructuras alternativas que se tuvieron en cuenta durante el desarrollo, pero que finalmente fueron descartadas.

También se explica cómo fueron implementados los métodos más relevantes, y por qué fueron implementados de esa manera.

En cuarto lugar, también se presentan las convenciones utilizadas para los métodos que involucran el pasaje de archivos, como el modify.

Finalmente, se presenta una conclusión acerca de las decisiones de diseño tomadas a lo largo del desarrollo del trabajo práctico.

**Instalación:**

Para instalar el trabajo práctico exitosamente, se debe ejecutar el comando *ant* en la carpeta principal, donde está ubicado el archivo *build.xml* (tener en cuenta que hay que tener ant instalado). El target default de *build.xml* generará dos carpetas, una llamada *dist*, y otra llamada *build*. En la carpeta *build* estarán todos los archivos *.class*, y en la carpeta *dist* se encuentra el archivo ejecutable *.jar*. Si se desea limpiar las carpetas generadas al ejecutar *ant*, el archivo *build.xml* también contiene un target llamado *clean*, que borrará las carpetas *build* y *dist*, por lo que sólo sería necesaria ejecutar el comando *ant clean*.

**Clase Blockchain:**

La clase Blockchain fue creada para modelar la Blockchain sobre la que se mantiene el árbol AVL.

A continuación se presenta una descripción de cómo fue pensada e implementada esta clase, y una justificación de las decisiones tomadas.

En primer lugar, se decidió modelar la Blockchain como una lista simplemente encadenada con header, con la particularidad de que cada elemento tiene una referencia al anterior, por lo que la lista se empieza a recorrer a partir del último elemento. En esta parte de la implementación, es posible analizar el siguiente punto de discusión: ¿Por qué una lista simplemente encadenada donde cada elemento apunta al anterior, y no al siguiente, o por qué no una lista doblemente encadenada? ¿Por qué una lista simplemente encadenada y no un ArrayList?.

Respecto a la primer pregunta, como veremos más adelante, el hecho de que cada elemento apunte al anterior, al siguiente, o a ambos, no modifica la complejidad temporal de los métodos implementados, por lo que se decidió elegir el modelo que le resultó más cómodo al equipo.

Respecto a la segunda pregunta, se requirió un poco más de reflexión para responderla. Sabemos que en un ArrayList las operaciones de acceso a un elemento son de orden O(1), ya que los elementos están contiguos en memoria y se acceden mediante índices. Pero, esto provoca que de vez en cuando, al insertar un elemento, se llegue al límite del tamaño fijo y haya que realocar el espacio donde están los elementos, provocando una inserción de orden O(n) en el peor caso. En cambio, al utilizar una lista simplemente encadenada, las operaciones de inserción siempre serán de orden O(1), ya que los elementos no están contiguos en memoria. El problema de esto es que las operaciones de acceso son de orden O(n). Es por esto que se tuvo que realizar una decisión sobre a qué darle más importancia: ¿Operaciones de acceso a un bloque de la Blockchain, u operaciones de inserción de bloques en la Blockchain? El único momento donde se realiza un acceso a un índice particular de la Blockchain es cuando se ejecuta el comando *modify* para modificar un bloque en específico. Pero cuando se modifica un bloque, lo más probable es que la Blockchain quede inválida, y si la Blockchain es inválida, ya no se podrá seguir usando porque no es confiable. En cambio, los comando como *add* y *remove* se ejecutarán con mucha frecuencia, ya que la idea de esta Blockchain es poder realizar operaciones sobre un árbol AVL. Como se realizarán más operaciones de inserción de bloques que de búsqueda de bloques, se optó por implementar una lista simplemente encadenada en vez de un ArrayList.

**Clase Block:**

La clase Block fue creada para modelar cada bloque de la Blockchain.