Informe parte práctica parcial 2

1. Descripción del Funcionamiento de los Árboles AVL y su Diferencia con los Árboles Binarios de Búsqueda:

* Los árboles AVL son un tipo de árbol de búsqueda binaria (BST), pero a diferencia de estos, mantienen un balance en su altura, la cual no puede diferir de 1 ni -1 para garantizar que esté correctamente balanceado. Esto permite que, al implementar todos sus métodos, como la inserción, eliminación y búsqueda, se tenga un tiempo de O(logn), en comparación con O(n) si no estuviera balanceado.

El funcionamiento de estos árboles es que cada vez que se inserta o se elimina un valor, el árbol se tiene que balancear automáticamente. Esto se hace verificando si sus alturas, como ya mencioné, no difieren de 1 y -1. Si llega a ocurrir esto, existen dos tipos de rotaciones para que el árbol pueda quedar balanceado: la rotación simple, que solo se rotaría a la izquierda o a la derecha, y la rotación doble, que se rotaría primero a la izquierda y luego a la derecha, o viceversa, realizando así dos rotaciones.

Además, este proceso tendrá que seguir un orden al rotar o insertar un valor. Por ejemplo, si se trata de valores numéricos, se organizaría de tal manera que, si el hijo es mayor que el padre, se coloca a la derecha; y si no, se coloca a la izquierda, y así sucesivamente con todos los nodos.

* La diferencia que tienen los árboles AVL y los árboles binarios de búsqueda (BST) es que, como mencioné anteriormente, los AVL tienen que estar balanceados y, para esto, requieren de rotaciones específicas para cumplir con esta condición en su altura. En cambio, los árboles BST no están necesariamente balanceados, lo que puede llevar a un mayor tiempo en la búsqueda, en la inserción y en la eliminación. Por ende, los árboles binarios de búsqueda serían más simples de implementar.

Por ejemplo, los árboles AVL son ideales para aplicaciones donde se requieren búsquedas rápidas y frecuentes, como en bases de datos y sistemas de archivos, donde el tiempo de respuesta tiene que ser rápido. En cambio, los árboles BST pueden ser más adecuados en situaciones donde se prefiere una implementación más sencilla y donde el rendimiento no es un factor crítico, como en pequeños conjuntos de datos o aplicaciones que no demandan un alto rendimiento.

2. Explicación de cómo se implementan las rotaciones para mantener el balance:

Para poder mantener el balance en el árbol, tenemos que implementar todos los tipos de rotación que hay. Por ende, en el código tenemos estos cuatro métodos:

* Rotación Izquierda: Esta es una rotación simple que se genera cuando un subárbol está desbalanceado hacia la derecha. Lo que hace es que el hijo izquierdo se convierte en el nuevo nodo raíz del subárbol.
* Rotación Derecha: Esta rotación se realiza cuando un nodo está desbalanceado hacia la izquierda. En este caso, el hijo derecho se convierte en la nueva raíz, funcionando de la misma manera que en la rotación anterior, pero en el sentido contrario.
* Rotación Izquierda Doble: Esta rotación se utiliza cuando hay un desbalance en el subárbol izquierdo del hijo derecho del nodo. Se realiza primero una rotación hacia la derecha en el hijo izquierdo, seguida de una rotación a la izquierda.
* Rotación Derecha Doble: Esta rotación es básicamente lo mismo que la rotación izquierda doble, pero en el sentido contrario. Se utiliza cuando hay un desbalance en el subárbol derecho del hijo izquierdo del nodo, haciendo la misma rotación del ejemplo anterior, pero hacia el lado contrario.

3. Análisis de la complejidad de las operaciones de inserción y eliminación en un Árbol AVL.

* Inserción: Para insertar un nuevo nodo lo primero que debemos hacer es encontrar la posición correcta del árbol en la cual lo podemos insertar, esto implica comparar el nuevo valor con los nodos ya existentes, empezando desde la raíz y bajando por los hijos, por el izquierdo si el valor es menor a la raíz o por el hijo derecho si el valor es mayor a la raíz y seguir así recorriendo, en el peor de los casos, todos los niveles del árbol hasta encontrar el sitio adecuado donde insertar el nuevo nodo.

Después es probable que se necesite reequilibrar el árbol, esto significa que toca revisar el balance de los nodos ancestros y, si es necesario, realizar las respectivas rotaciones para mantener el equilibrio

* Eliminación: para eliminar un nodo lo primero que tenemos que hacer es recorrer el árbol para buscar el nodo a eliminar, esto se hace de la misma manera que a la hora de la inserción, comparando el valor que se necesite eliminar con los nodos ya existentes.

Después de verificar que este nodo existe, se elimina y al ser un árbol AVL puede ser necesario ajustar algunos nodos para mantener el equilibrio del árbol, esto incluye encontrar un nodo sustituto si el nodo a eliminar tiene dos hijos y luego aplicar las rotaciones necesarias.

Bibliografías

* Para la implementación de este código nos guiamos de este video de YouTube:
* [Arbol AVL Java (youtube.com)](https://www.youtube.com/watch?v=8PRXJRsg-Mw)
* Además, sacamos información de las siguientes páginas
* [Guide to AVL Trees in Java | Baeldung](https://www.baeldung.com/java-avl-trees)
* [Árbol AVL \_ AcademiaLab (academia-lab.com)](https://academia-lab.com/enciclopedia/arbol-avl/)
* [EstructuraDatosAlgoritmos1/S07\_EDNoLineales\_Arboles.ipynb at main · carlosalvarezh/EstructuraDatosAlgoritmos1 (github.com)](https://github.com/carlosalvarezh/EstructuraDatosAlgoritmos1/blob/main/S07_EDNoLineales_Arboles.ipynb)

* Además para la complementación del método eliminar nos apoyamos de chat gpt, ya que, no teníamos conocimientos de la implementación de este, además nos ayudó al entendimiento de este método.