**Memoria Práctica 2 Estructura de Datos**

****

Trabajo realizado por:

 Sergio Sanchiz Villena - [sergio.sanchiz@goumh.umh.es](mailto:sergio.sanchiz@goumh.umh.es)

Clase Nodo y Clase ABB

La clase **Nodo** es una clase que representa un nodo en un árbol binario de búsqueda. Cada nodo tiene un valor (valor\_objeto), que puede ser un objeto cualquiera, y un identificador (id), que se utiliza para ordenar los nodos en el árbol. Además, cada nodo tiene dos referencias a otros nodos, uno para el hijo izquierdo (izquierdo) y otro para el hijo derecho (derecho). Si el nodo no tiene hijos, las referencias a los hijos serán None.

La clase **ArbolBinarioBusqueda** es una clase que representa un árbol binario de búsqueda. Cada árbol tiene una raíz (raiz), que es un objeto de la clase Nodo y representa el nodo superior del árbol. Si el árbol está vacío, la referencia a la raíz será None.

El árbol binario de búsqueda es una estructura de datos en la que cada nodo tiene como máximo dos hijos, y cada nodo izquierdo tiene un valor menor que el nodo padre, mientras que cada nodo derecho tiene un valor mayor que el nodo padre.

El árbol tiene varias funciones para manipular los nodos y el árbol en sí mismo:

* **insertar(id, valor)**: Esta función inserta un nuevo nodo en el árbol con un identificador id y un valor valor. El nodo se coloca en su posición correspondiente en el árbol según el valor del identificador. Si un nodo con el mismo identificador ya existe en el árbol, su valor se reemplaza por el valor pasado a la función.
* **buscar(**valor\_buscar**)**: Esta función busca un nodo en el árbol cuyo valor sea igual al valor pasado como argumento. Si el nodo se encuentra, se devuelve el objeto que representa el valor del nodo. Si no se encuentra el nodo, se devuelve None.
* **eliminar(id)**: Esta función elimina el nodo con el identificador **id** del árbol. Si el nodo tiene hijos, se reorganiza el árbol para que el nodo eliminado sea reemplazado por el nodo hijo correspondiente.
* **recorrido\_inorden(**nodo**)**: Esta función realiza un recorrido en orden (in-order) del árbol, comenzando desde el nodo pasado como argumento. En un recorrido in-order, primero se visitan los nodos del subárbol izquierdo, luego el nodo actual y finalmente los nodos del subárbol derecho.
* **recorrido\_preorden**(nodo): Esta función realiza un recorrido en preorden (pre-order) del árbol, comenzando desde el nodo pasado como argumento. En un recorrido pre-order, primero se visita el nodo actual, luego los nodos del subárbol izquierdo y finalmente los nodos del subárbol derecho.
* **recorrido\_postorden**(nodo): Esta función realiza un recorrido en postorden (post-order) del árbol, comenzando desde el nodo pasado como argumento. En un recorrido post-order, primero se visitan los nodos del subárbol izquierdo, luego los nodos del subárbol derecho y finalmente el nodo actual.
* **recorrido\_anchura**(nodo): Esta función realiza un recorrido por anchura (breadth-first) del árbol, comenzando desde el nodo pasado como argumento.
* **eliminar**(id) Esta función elimina un nodo del árbol binario de búsqueda. Si el nodo tiene hijos, se utiliza el método del sucesor o del predecesor para mantener la propiedad de árbol binario de búsqueda. También se elimina el cliente correspondiente a través de la función eliminar\_cliente\_por\_id().
* **buscar\_nodo**(id) Esta función busca un nodo en el árbol binario de búsqueda con el id proporcionado y devuelve el nodo si lo encuentra, de lo contrario, devuelve None.
* **eliminar\_nodo**(id) Esta función elimina un nodo del árbol binario de búsqueda. Si el nodo tiene hijos, se utiliza el método del sucesor o del predecesor para mantener la propiedad de árbol binario de búsqueda. Sin embargo, en este caso no se elimina el cliente correspondiente.

Problemas encontrados

A la hora de hacer el programa lo más complicado que he tenido ha sido a la hora de eliminar el cliente y el nodo de un árbol, ya que, en un inicio tenía pensado usar una lista que contenía la información del cliente.

Pero a la hora de hacer la comparación con los nodos hijos para agruparlos cuando el nodo padre se elimine, siempre me salía el error de la imagen



Ya que no comprendía ese error porque teóricamente comparaba números (id del nodo y de la lista cliente) pero al ser de distintos tipos de variables, el programa no funcionaba, entonces cambie la orientación del programa y lo hice con una clase y objeto cliente.

Por lo que era más fácil sacar los valores de la identidad del nodo, y del objeto cliente y pasarlos a int para poder compararlos, salvo eso y que no he logrado hacer el recorrido IN\_ORDER en la simulación con solo las IDs de los clientes, lo demás he podido lograrlo sin mayor problema.