Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2023-2024

Práctica 7: Implementación del TDA AVL

1. Objetivo

El objetivo de esta práctica es trabajar con el TDA árbol binario de búsqueda balanceado (AVL) [1][3], realizar una implementación de sus algoritmos y comprobar de forma empírica la complejidad computacional del TDA. La implementación en lenguaje C++ utiliza la definición de tipos genéricos (plantillas), el polimorfismo dinámico y la sobrecarga de operadores.

2. Entrega

Se realizará en una sesión de laboratorio en las siguientes fechas:

Sesión de entrega: del 29 de abril al 3 de mayo

3. Enunciado

Se denomina árbol binario (AB) a un árbol de grado 2 [5].

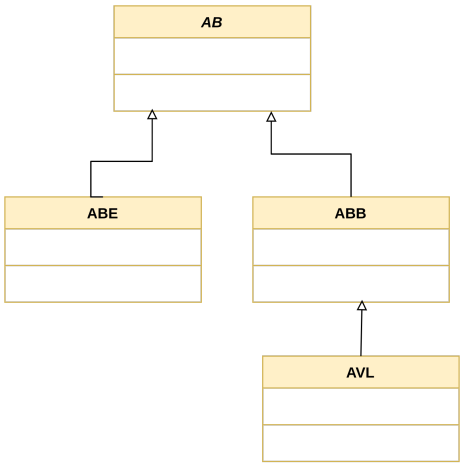
Un árbol binario de búsqueda (ABB) [6] es un AB en el que se cumple que el valor de la raíz de cada rama es:

- Mayor que los valores de los nodos de su subárbol izquierdo.
- Menor que los valores de los nodos de su subárbol derecho.

Un árbol binario de búsqueda balanceado (AVL) es un ABB en el que la diferencia de las alturas de los dos subárboles de cada rama no supera la unidad. Cuando se inserta o elimina un nodo del árbol puede producirse un desbalanceo si la diferencia de las alturas de los dos subárboles llega a ser igual a dos. En estos casos hay que rebalancear utilizando rotaciones: Izquierda-Izquierda (II), Derecha-Derecha (DD), Izquierda-Derecha (ID) o Derecha-Izquierda (DI).

Las operaciones que se pueden realizar en el `AVL<Key>` son las mismas que en cualquier otro árbol binario.

Se pide actualizar la jerarquía implementada en la práctica 6 [3]: a partir `ABB<Key>` se deriva el tipo de dato genérico `AVL<Key>`.



4. Notas de implementación

La implementación del TDA AVL se realiza teniendo en cuenta las siguientes consideraciones:

1. Para representar los nodos de un AVL se utiliza la clase genérica `NodoAVL<Key>` derivada de la clase genérica `NodoB<Key>` (definida en el enunciado de la práctica 6 [3]). La clase genérica `NodoAVL<Key>` contiene el siguiente atributo adicional:
    a. Atributo privado `bal`, que contiene el factor de balanceo del nodo.
2. A partir de la clase `ABB<Key>` se deriva la clase `AVL<Key>` que redefine el método para insertar un valor en el árbol. La clase `AVL<Key>` **no** admite la inserción de valores repetidos. En el caso de que se intente insertar un valor que ya se encuentra en el árbol, el método `insertar` retorna el valor `false`.
3. Para permitir observar las operaciones realizadas en un AVL cuando se produce un desbalanceo se va a incluir un "modo traza" en la ejecución:
    a. En un AVL, cuando el modo traza esté activado y se produzca un desbalanceo, se mostrará por pantalla el árbol antes de aplicar la rotación, el tipo de rotación que se va a aplicar (II, DD, ID, DI) y en qué nodo. Además, en cada nodo no vacío del AVL se mostrará, entre paréntesis, el balanceo del nodo.
    b. En la impresión de un ABE o un ABB no hay diferencia entre el modo traza activado o desactivado.
4. El programa principal acepta las siguientes opciones por línea de comandos:
    a. `-ab <abe|abb|avl>`, para indicar el tipo de árbol con el que se va a trabajar.
    b. `-init <i> [s][f]`, indica la forma de introducir los datos de la secuencia

        ```
        i=manual
        i=random [s], s  es el número de elementos a generar
        i=file [s][f], s  es el número de elementos a generar
                          f  es nombre del fichero de entrada
        ```

    c. `-trace <y|n>`, indica si se muestra o no la traza durante la ejecución.
5. Se utiliza la clase `nif` definida en la práctica 4 [4] como tipo de dato `Key` en la plantilla .
6. El tipo de dato `nif` se actualiza, si es necesario, con la sobrecarga de los operadores que garanticen el correcto funcionamiento del árbol.
7. Se crea un árbol según los parámetros recibidos por el programa. Si se elige la opción de inicialización manual se genera un árbol vacío. En otro caso, se inicializa el árbol con los valores adecuados.
8. Se presenta un menú con las siguientes opciones:

    ```
    [0] Salir
    [1] Insertar clave
    [2] Buscar clave
    [3] Mostrar árbol inorden
    ```

9. Para cada operación de inserción o búsqueda se solicita el valor de clave y se realiza la operación.

10. Después de cada operación de inserción se muestra el árbol resultante mediante el recorrido por niveles, utilizando la sobrecarga del operador.

Ejemplo de visualización del `AVL<long>` generado sin traza:

Árbol vacío
Nivel 0: [.]

Insertar: 30
Nivel 0: [30]
Nivel 1: [.] [.]

Insertar: 25
Nivel 0: [30]
Nivel 1: [25] [.]
Nivel 2: [.] [.]

Insertar: 15
Nivel 0: [25]
Nivel 1: [15] [30]
Nivel 2: [.] [.] [.] [.]

Insertar: 40
Nivel 0: [25]
Nivel 1: [15] [30]
Nivel 2: [.] [.] [.] [40]
Nivel 3: [.] [.]

Ejemplo de visualización del `AVL<long>` generado con traza:

Árbol vacío
Nivel 0: [.]

Insertar: 30
Nivel 0: [30(0)]
Nivel 1: [.] [.]

Insertar: 25
Nivel 0: [30(1)]
Nivel 1: [25(0)] [.]
Nivel 2: [.] [.]

Insertar: 15
Desbalanceo:
Nivel 0: [30(2)]
Nivel 1: [20(1)] [.]
Nivel 2: [15(0)] [.] [.] [.]
Nivel 3: [.] [.]
Rotación II en [30(2)]:
Nivel 0: [25(0)]
Nivel 1: [15(0)] [30(0)]
Nivel 2: [.] [.] [.] [.]

Insertar: 40
Nivel 0: [25(-1)]
Nivel 1: [15(0)] [30(-1)]
Nivel 2: [.] [.] [.] [40(0)]
Nivel 3: [.] [.]

5. Referencias

[1] Google: Apuntes de clase.
[2] Wikipedia: Árbol binario de búsqueda balanceado: https://es.wikipedia.org/wiki/Árbol_AVL
[3] Aula virtual AyEDA: Práctica 6
[4] Aula virtual AyEDA: Práctica 4
[5] Wikipedia: Árbol binario: https://es.wikipedia.org/wiki/Árbol_binario
[6] Wikipedia: Árbol binario de búsqueda: https://es.wikipedia.org/wiki/Árbol_binario_de_búsqueda