Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2024-2025

Práctica 6: Implementación del TDA Árbol

1. Objetivo

El objetivo de esta práctica es trabajar con los TDA árbol binario de búsqueda (ABB) y árbol binario de búsqueda balanceado (AVL) [1][2][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 dos sesiones de laboratorio en las siguientes fechas:

Sesión tutorada: del 21 al 24 de abril
Sesión de entrega: del 28 al 30 de abril*

* El alumnado afectado por la festividad del 1 de mayo podrá asistir a la sesión de prácticas en cualquier otro grupo

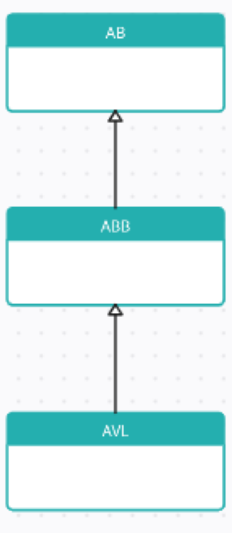
3. Enunciado

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

Un árbol binario de búsqueda (ABB) 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).



Se pide definir el tipo de dato genérico abstracto `AB<Key>` que permite representar cualquier árbol binario para contener valores de tipo clave, `Key`. A partir de `AB<Key>` se derivan los tipos de datos genéricos `ABB<Key>` y `AVL<Key>`.

La implementación del tipo genérico debe permitir realizar al menos las siguientes operaciones:

- Buscar: Se comprueba si una clave `key` dada se encuentra en el árbol.
- Insertar: Se añade una nueva clave `key` al árbol.
- Recorrer: Se recorren todos los nodos del árbol. Según el orden establecido en el recorrido podemos encontrar, entre otros, los siguientes:

- Inorden: Se recorre en orden: subárbol izquierdo - raíz - subárbol derecho
- Por niveles: Se recorren los nodos de los diferentes niveles del árbol en orden creciente desde la raíz y dentro de cada nivel de izquierda a derecha.

4. Notas de implementación

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

1. Para implementar los nodos de un árbol binario se define una clase genérica `NodoB<Key>` con los siguientes atributos:
   a. Atributo protegido `dato`, de tipo `Key`, que contiene la información a almacenar en los nodos del árbol.
   b. Atributo protegido `izdo`, es un puntero a la propia clase `NodoB<Key>` y representa el hijo izquierdo del nodo binario.
   c. Atributo protegido `dcho`, es un puntero a la propia clase `NodoB<Key>` y representa el hijo derecho del nodo binario.
2. Se define la clase genérica abstracta `AB<Key>` para representar un árbol binario. Esta clase contiene un atributo, `raiz` (puntero a la clase `NodoB<Key>`). Si el árbol está vacío este atributo tendrá asignado el valor `nullptr`.
3. En esta clase genérica abstracta `AB<Key>` se definen los siguientes métodos:
   a. Método público nulo `bool insertar(const Key& k)`: retorna el valor booleano `true` si se inserta el valor `k` del tipo `Key` en el árbol. En otro caso se retorna el valor booleano `false`.
   b. Método público nulo `bool buscar(const Key& k) const`: retorna el valor booleano `true` si el valor `k` del tipo `Key` está almacenado en el árbol. En otro caso, retorna el valor `false`.
   c. Método público `void inorden( ) const`: realiza un recorrido inorden del AB mostrando los nodos por pantalla.
   d. Sobrecarga del operador de inserción en flujo para mostrar el `AB<Key>` utilizando el recorrido por niveles: En cada nivel se muestran los nodos de izquierda a derecha. El subárbol vacío se visualiza con `[.]`.
4. A partir de la clase `AB<Key>` se deriva la clase `ABB<Key>` para representar el árbol binario de búsqueda. La clase `ABB<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`.
5. Para representar los nodos de un AVL se utiliza la clase genérica `NodoAVL<Key>` derivada de la clase genérica `NodoB<Key>`. La clase genérica `NodoAVL<Key>` contiene el siguiente atributo adicional:
   a. Atributo privado `bal`, que contiene el factor de balanceo del nodo.

6. 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`.

7. 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 ABB no hay diferencia entre el modo traza activado o desactivado.

8. El programa principal acepta las siguientes opciones por línea de comandos:

    a. `-ab <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 taza durante la ejecución.

9. Se utiliza la clase `nif` definida en la práctica 4 [4] como tipo de dato `Key` en la plantilla .

10. El tipo de dato `nif` se actualiza, si es necesario, con la sobrecarga de los operadores que garanticen el correcto funcionamiento del árbol.

11. 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.

12. Se presenta un menú con las siguientes opciones:

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

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

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

Ejemplo de visualización de un `ABB<long>`:

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: [30]
Nivel 1: [25] [.]
Nivel 2: [15] [.]
Nivel 3: [.] [.]

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

Ejemplo de visualización del `AVL<long>` generado con el “modo traza” no activado:

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 el “modo traza” activado:

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:
https://es.wikipedia.org/wiki/Árbol_binario_de_búsqueda
[3] Wikipedia: Árbol binario de búsqueda balanceado:
https://es.wikipedia.org/wiki/Árbol_AVL
[4] Aula virtual AyEDA: Práctica 4