Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2022-2023

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] y realizar una implementación en lenguaje C++ de la estructura de datos y sus algoritmos. Se utilizará la definición de tipos genéricos, el polimorfismo dinámico y la sobrecarga de operadores.

2. Entrega

Se realizará en dos sesiones de laboratorio en las siguientes fechas:

Sesión tutorada: del 24 al 28 de abril de 2023
Sesión de entrega: del 2 al 8 de mayo de 2023

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 deriva los tipos de datos genéricos `ABB<Key>` y `AVL<Key>`. Implementar un programa en C++ que permita observar su funcionamiento.

Cualquier árbol binario debe permitir realizar al menos las operaciones:

- Insertar: Se añade una nueva clave `key` al árbol.
- Buscar: Se comprueba si una clave `key` dada se encuentra en el 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

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 el 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>` que representa el árbol binario de búsqueda. La clase `ABB<Key> no` admitirá 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. A partir de la clase `ABB<Key>` se deriva la clase `AVL<Key>` que representa el árbol binario de búsqueda balanceado y que redefine el método para insertar un nodo en el árbol.

6. Para representar los nodos de un AVL se utiliza la clase genérica `NodoAVL<Key>` derivada de la clase genérica `NodoB<Key>`, que contiene el siguiente atributo adicional:

    a. Atributo privado `bal`, que contiene el factor de balanceo del nodo.

7. Para permitir observar las operaciones realizadas en un AVL cuando se produce un

desbalanceo se va a incluir un “modo traza” en la compilación. Se utilizará una etiqueta que indicará al precompilador si tiene que incluir o excluir el código de visualización en el ejecutable.

   a. En un AVL generado en modo traza, cuando 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.

8. El programa principal tiene el siguiente comportamiento:

   a. Se utilizará `Key = long` para realizar la ejecución del programa.

   b. Se pide al usuario el tipo de árbol binario que quiere crear (`ABB<Key>` o `AVL<Key>`).

   c. Se genera un árbol vacío.

   d. Se presenta un menú con las siguientes opciones:

```
[0] Salir

[1] Insertar clave

[2] Buscar clave

[3] Mostrar árbol inorden
```

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

   f. 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 `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 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 traza:

Árbol vacío
Nivel 0: [.]

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

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

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

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

9. Durante las sesiones de laboratorio se podrán solicitar cambios y/o ampliaciones en la especificación de este enunciado.

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