Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2021-2022

Práctica 6: Implementación de árboles binarios equilibrados

1. Objetivo

El objetivo de esta práctica es trabajar con la estructura de datos árbol binario equilibrado [1][2] y realizar una implementación en lenguaje C++ de la estructura de datos y sus algoritmos. Se utilizará la definición de tipos genéricos y la sobrecarga de operadores.

2. Entrega

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

Sesión tutorada: del 2 al 6 de mayo de 2022
Sesión de entrega: del 9 al 13 de mayo de 2022

3. Enunciado

Un árbol binario (AB) está equilibrado si la diferencia de los tamaños de sus dos subárboles es como máximo uno. Un árbol binario está perfectamente equilibrado si están equilibradas todas sus ramas. Utilizaremos el término árbol binario equilibrado (ABE) para referirnos a un AB perfectamente equilibrado.

Se pide desarrollar un tipo de dato genérico que representa un árbol binario equilibrado `ABE<Key>` para contener valores de tipo clave, `Key`, y realizar un programa en C++ que permita observar su funcionamiento. Cualquier árbol binario `AB<Key>` debe permitir realizar 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 privado `dato`, de tipo `Key`, que contiene la información a almacenar en el árbol.
    b. Atributo privado `izdo`, es un puntero a la propia clase `NodoB<Key>` y representa el hijo izquierdo del nodo binario.
    c. Atributo privado `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 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 `ABE<Key>` que representa el árbol binario equilibrado e implementa los métodos nulos definidos en `AB<Key>` garantizando la inserción equilibrada en el árbol. La clase `ABE<Key>` no admitirá la inserción de valores repetidos.

5. El programa principal tiene el siguiente comportamiento:

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

    b. Se genera un `ABE<Key>` vacío.

    c. Se presenta un menú con las siguientes opciones:

```
[0] Salir

[1] Insertar clave

[2] Buscar clave

[3] Mostrar árbol inorden
```

    d. Para cada inserción o búsqueda se solicita el valor de clave y se realiza la operación en el `ABE<Key>`.

    e. Después de cada operación de inserción, se muestra el `ABE<Key>` resultante mediante el recorrido por niveles, utilizando la sobrecarga del operador.

Ejemplo de visualización del `ABE<int>`:

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] [15]
Nivel 2: [.] [.] [.] [.]

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

6. 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: https://es.wikipedia.org/wiki/Árbol_binario