Título del epígrafe

Árboles binarios de búsqueda.

Descripción del epígrafe

En este epígrafe se estudiarán otras estructuras arbóreas, tales como: Árboles binarios de búsqueda (Árboles AVL, Árboles rojinegros, Árboles AA).

Descriptores del epígrafe

Árboles binarios de búsqueda.

Conceptos clave

Un ***árbol binario de búsqueda*** es un árbol binario que, recorrido en orden simétrico, permite obtener la información de los nodos en algún criterio de ordenamiento.

Árboles binarios de búsqueda

Como se ha estudiado hasta ahora, una de las operaciones que más realizamos sobre cualquier estructura es la búsqueda y por esto es tan importante hacer implementaciones eficientes.

Para un volumen muy grande de datos, la búsqueda lineal es prohibitiva. Esto puede mejorarse con los árboles, teniendo en cuenta que se pueden reducir las búsquedas a subárboles y por tanto, el tiempo de ejecución de muchas de sus operaciones es O(log N), aunque en el peor caso es O(N).

Un **árbol binario de búsqueda** es un árbol binario que, recorrido en orden simétrico, permite obtener la información de los nodos en algún criterio de ordenamiento.

Es importante señalar que en el libro de texto solo tratan el orden ascendente.

La técnica de construcción de este árbol consiste en un proceso recursivo que va colocando los nodos en el subárbol izquierdo o derecho del nodo raíz, según sea el criterio de ordenamiento deseado, o sea, ascendentemente o descendentemente.

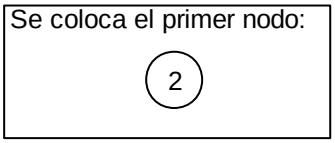
Si el criterio considera el ordenamiento de manera ascendente, el nodo que se quiere insertar se compara con la raíz del árbol. Si es menor se coloca en el subárbol izquierdo siguiendo el mismo proceso. En caso contrario, se coloca en el subárbol derecho siguiendo el mismo proceso.

A modo de ejemplo, supongamos que el criterio de ordenamiento es ascendente y que se quiere construir el árbol binario de búsqueda correspondiente a la siguiente lista de números:

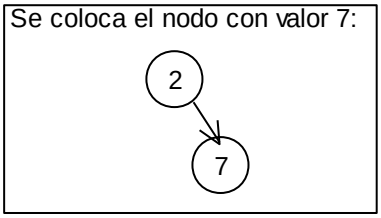
2, 7, 1, 4, 5

PROFESOR: Hacer una animación para este ejemplo y que se muestre como se van insertando los nodos en el árbol, tal y como se explica en este texto.

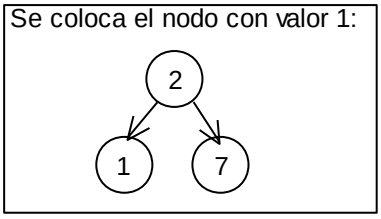
Se parte del árbol vacío y se coloca el valor 2.



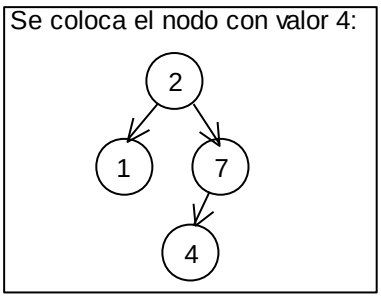
Ahora hay que colocar el valor 7, pero en la rama derecha del árbol, ya que es mayor que la raíz.



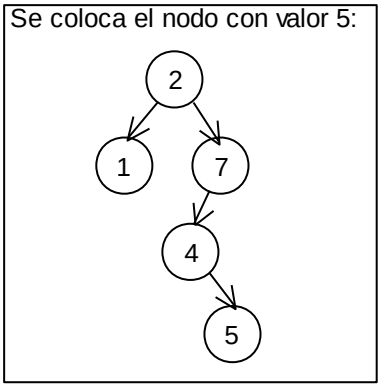
El valor 1 se compara con el 2 y, como es menor, este nodo se coloca en el subárbol izquierdo del 2.



El próximo valor a colocar es el 4, por lo que se compara con el 2 y, al ser mayor, se debe colocar en el subárbol derecho, pero en el subárbol derecho hay un nodo, luego se repite el proceso con ese nodo y, como 4 es menor que 7, se coloca este nodo en el subárbol izquierdo.



El valor 5 es mayor que 2, por lo que debe ir en el subárbol derecho del nodo raíz, es menor que 7, por lo que se debe colocar en el subárbol izquierdo de este nodo y es mayor que 4, por lo que se debe insertar en la rama derecha de este último nodo.

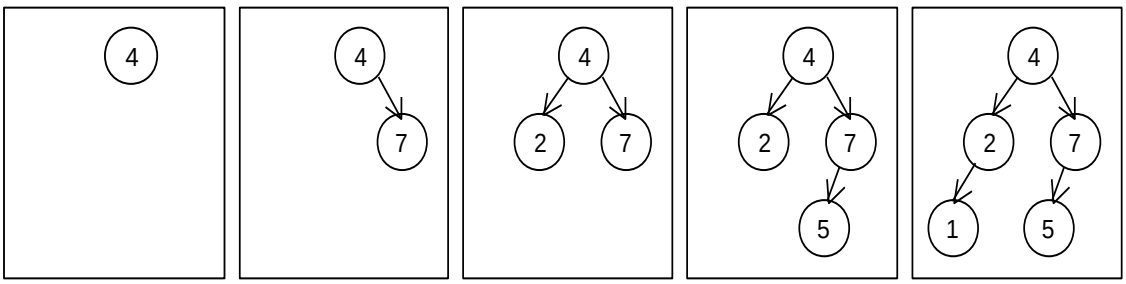


Una vez construido el árbol, se puede realizar un recorrido en orden simétrico y se obtiene la información de sus nodos en orden ascendente: 1, 2, 4, 5, 7.

Es necesario aclarar que si se cambia el orden de los valores de la lista original se obtiene un árbol distinto al anterior, pero el efecto del recorrido en orden simétrico devuelve la lista ordenada ascendentemente.

Pongamos, por ejemplo, otro orden de los valores de la lista original. Digamos: 4, 7, 2, 5, 1. El árbol lexicográfico para esta nueva lista se construye siguiendo los siguientes pasos:

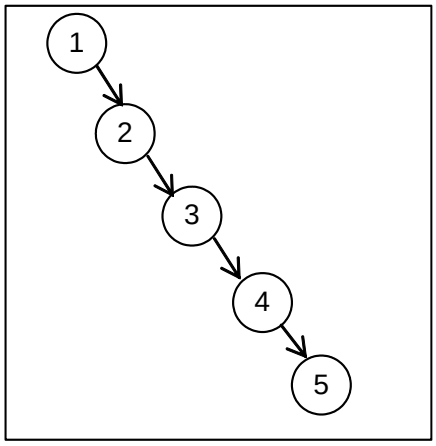
Hacer animación aquí



Ahora, un recorrido en orden simétrico de este árbol da como resultado la lista: 1, 2, 4, 5, 7.

Estos árboles tienen muchas aplicaciones en el campo de la Informática, pues constituyen un mecanismo relativamente simple y rápido para ordenar colecciones de informaciones.

En ocasiones la búsqueda en un árbol binario de búsqueda puede convertirse en una búsqueda secuencial. Por ejemplo, considere un árbol como el representado en la figura siguiente:



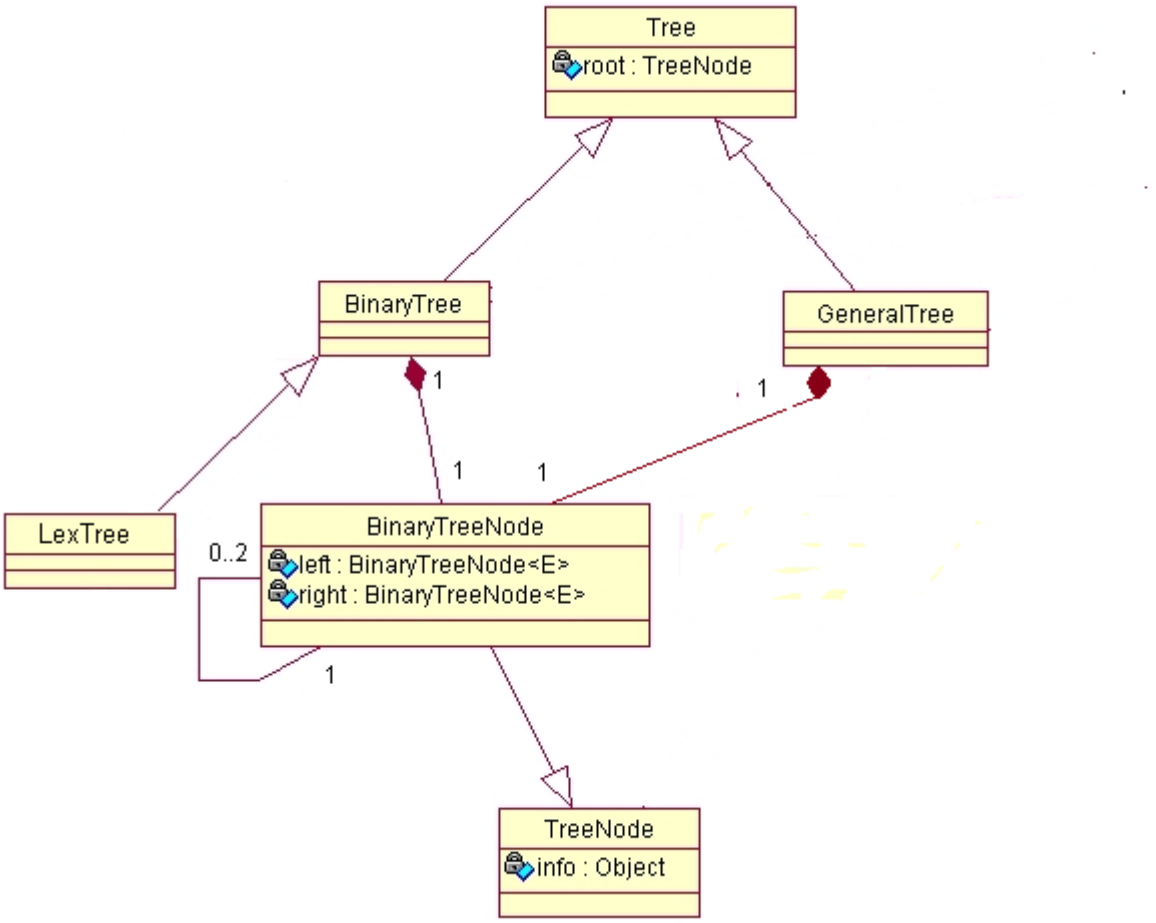
En este caso para buscar el 5 habría que recorrer todo el árbol. Esto sucede porque **el árbol no está balanceado**, es decir los nodos no están distribuidos uniformemente y se han insertado todos los nodos en profundidad (ejemplo, considerar que el orden de inserción de los nodos para el gráfico fue 1, 2, 3, 4 y 5). Esto podría ser salvado si se utilizara un árbol que tuviese en cuenta en el momento de la inserción la cantidad de niveles que tiene el árbol y cómo insertar de forma que se distribuyan los nodos uniformemente.

Una solución a este problema es la condición estructural adicional de equilibrio. Esto implica que sean árboles más complicados, sobre todo las inserciones y eliminaciones requieren mantener dicho equilibrio, pero las búsquedas y los accesos son más eficientes.

Dentro de este tipo de árboles equilibrados se encuentran los árboles AVL, los árboles rojinegros, los árboles AA y los árboles biselados. El alumno puede estudiar estos tipos de árboles de búsqueda en el libro de texto.

Implementación de los árboles binarios de búsqueda

Estamos en presencia de tipos específicos de árboles binarios, que modifican las operaciones de inserción y eliminación de forma tal que la información contenida en los nodos esté ordenada si se recorre el árbol en simétrico.
Por tanto la implementación puede ser una clase que herede de la clase BinaryTree, y que redefina los métodos de inserción y eliminación de acuerdo a las exigencias de cada tipo de árbol. El diagrama de clases UML que se presenta a continuación contiene la jerarquía descrita.




```
public class LexicographicTree<E> extends BinaryTree<E>{
        public enum Order {ASC, DESC};

        private Order order;

        public LexicographicTree();
        public LexicographicTree(Order order);
```

```
    public LexicographicTree(Order order, E rootInfo) throws
DoesNotImplementsComparable;

    public E deleteNode(E info);

    public void insertValue(E info) throws DoesNotImplementsComparable;
    private void insertInTree(BinaryTreeNode<E> root, BinaryTreeNode<E> node);
    private void insertValueIterative(E info) throws DoesNotImplementsComparable;
    public boolean insertNode(BinaryTreeNode<E> node, char type,
BinaryTreeNode<E> father);

    public LinkedList<E> getOrderedItems();

    private boolean implementsComparable(Object object);

}
```

SEUDOCÓDIGO del algoritmo insertar un nodo: (ITERATIVO)

Si el árbol está vacío insertar el nodo como raíz
Sino
    cursor = primero en el recorrido en simétrico
    MIENTRAS cursor < nodo a insertar y queden nodos por visitar **HACER**
        anterior = cursor
        cursor = siguiente en simétrico
    Fin MIENTRAS
    Si cursor no es nulo y cursor no tiene hijo izquierdo
        Insertar el nuevo nodo en el enlace izquierdo de cursor
    Sino
        Insertar el nuevo nodo en el enlace derecho de anterior
    FinSino

FinSino
FIN

Destacar que en este tipo de árbol la inserción se hace ordenada y por tanto se hace necesario recorrerlo en simétrico para obtener la posición que le corresponde al nodo que se pretende insertar.