Algorítmica II
Ingeniería Informática en Sistemas de Información
EPD-5: ÁRBOLES I

Introducción

En esta EPD se explorarán los conceptos fundamentales de árboles, profundizando en el ámbito de los árboles binarios y los árboles binarios de búsqueda. Inicialmente se presentarán los Tipos Abstractos de Datos (TAD) de cada uno de ellos, que deberán ser usados para resolver los experimentos y problemas propuestos en esta práctica. Una vez conocidos los TAD, se expondrá una implementación clásica de los árboles binarios y de los árboles binarios de búsqueda basadas en la configuración jerárquica de nodos enlazados.

Árboles

Un árbol (*tree*) es una estructura no lineal en la que los elementos están organizados en una jerarquía. Está compuesto de un conjunto de *nodos* en los que se almacenan los elementos y *aristas* que conectan unos nodos con otros. Cada nodo se encuentra en un *nivel* concreto de la jerarquía del árbol, siendo la *raíz* el único nodo situado en el nivel superior del árbol. Los nodos situados en los niveles inferiores son los *hijos* de los nodos ubicados en el anterior nivel. Cada nodo solo puede tener un *padre*, pero un nodo puede tener múltiples hijos. Los nodos que tienen el mismo padre se denominan *hermanos*. Evidentemente el nodo raíz es el único del árbol que no tiene padre. Un nodo que no tenga ningún hijo se denomina *hoja*, del mismo modo, un nodo que no sea la raíz y tenga al menos un hijo se le llama nodo *intermedio*.

En la figura 1 podemos ver que el nodo A es la raíz, los nodos C, D, E y F son nodos hoja y el nodo B es intermedio.

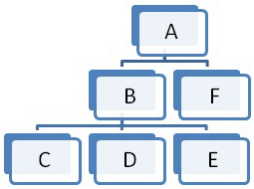

Figura 1. Ejemplo de árbol.

La raíz es el punto de entrada de un árbol. Podemos seguir un *camino* a través del árbol desde un padre a un hijo. Por ejemplo, en la figura 1, el camino que va desde el nodo A al D está formado por A, B, D. Un nodo es *ascendiente* de otro nodo si está situado por encima suyo en el camino que une ese nodo con la raíz del árbol. Por tanto, la raíz es un ascendiente de todos los nodos del árbol. Los nodos que pueden alcanzarse siguiendo un camino que parte de uno concreto se denominan *descendientes* de dicho nodo.

El *nivel* de un nodo es también la *longitud del camino* que va desde la raíz hasta ese nodo. Esta longitud de camino se determina contando el número de aristas que hay que seguir para llegar desde la raíz hasta el nodo. La raíz se considera nivel 0, los hijos de la raíz nivel 1, los nietos de la raíz nivel 2, y así sucesivamente. La *altura* de un árbol es la longitud del camino más largo que une la raíz con una hoja. Volviendo al ejemplo, el árbol de la figura 1 tiene una altura de valor 2.

Los árboles pueden clasificarse de muchas formas distintas. El criterio más importante es el número máximo de hijos que cada nodo del árbol pueda tener. Este valor se suele denominar *orden* del árbol. Un árbol que no tenga ningún límite en lo que respecta al número de hijos que un nodo puede tener se denomina *árbol general*. Un árbol que limite cada nodo a no más de *n* hijos se denomina *árbol n-ario*. De estos hay uno de especial importancia que son los *árboles binarios*, en cuyo Tipo Abstracto de Datos (TAD) nos centraremos en esta práctica.

Árboles Binarios

Un *árbol binario* es una estructura de datos en la cual cada nodo tiene como máximo dos nodos hijos. Típicamente los nodos hijos son llamados *izquierdo* y *derecho*. En la figura 2 podemos ver un ejemplo de árbol binario.

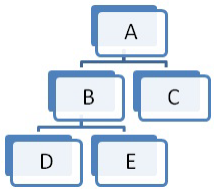

Figura 2. Árbol Binario.

Como con cualquier tipo de estructuras de datos se necesita contar con algoritmos para recorrer su contenido. Existen una serie de recorridos que se llevarán a cabo frecuentemente y son aquellos que pasan por todos los nodos de un árbol. Este tipo de recorridos se pueden clasificar en: *recorridos en profundidad* y *recorridos en anchura* o *por niveles*.

En cuanto a los recorridos en profundidad, podemos distinguir:

1) *Recorrido en preorden*: consiste en visitar el nodo actual y después visitar el subárbol izquierdo y después el subárbol derecho. En el ejemplo de la figura 2: A B D E C.

2) *Recorrido en inorden*: primero se visita el subárbol izquierdo, después el nodo actual y seguidamente el subárbol derecho. En el ejemplo de la figura 2: D B E A C.

3) *Recorrido en postorden*: se visita en primer lugar el subárbol izquierdo, después el subárbol derecho y finalmente el nodo actual. En el ejemplo de la figura 2: D E B C A.

El recorrido por niveles se puede realizar de múltiples maneras, pero la más extendida consiste en empezar por la raíz e ir descendiendo por los distintos niveles recorriendo de izquierda a derecha todos los nodos de cada nivel. Volviendo al ejemplo de la figura 2: A B C D E.

Para trabajar con este tipo de colecciones se usará el TAD Árbol Binario (*BinaryTreeADT*) que tendrá disponible el conjunto de operaciones representadas en la tabla1, que servirán para resolver cualquier problema donde pueda ser útil el uso de un árbol binario para implementar la solución.

Tabla 1. Operaciones comunes de los Árboles Binarios.

Operación	Descripción
`removeLeftSubtree`	Elimina el subárbol izquierdo de la raíz
`removeRightSubtree`	Elimina el subárbol derecho de la raíz
`removeAllElements`	Elimina todos los elementos del árbol
`isEmpty`	Determina si el árbol está vacío
`size`	Determina el número de elementos del árbol
`contains`	Determina si el elemento especificado está contenido en el árbol
`find`	Devuelve una referencia al elemento especificado, si se encuentra
`toString`	Devuelve una representación del árbol en forma de cadena de caracteres
`iteratorInOrder`	Devuelve un iterador para el recorrido del árbol en inorden
`iteratorPreOrder`	Devuelve un iterador para el recorrido del árbol en preorden
`iteratorPostOrder`	Devuelve un iterador para el recorrido del árbol en postorden
`iteratorLevelOrder`	Devuelve un iterador para el recorrido por niveles del árbol

Los árboles binarios pueden ser implementados usando múltiples estructuras de datos. En concreto para esta práctica usaremos la implementación clásica basada en nodos enlazados representada en la figura 3 (*LinkedBinaryTree)*.

Figura 3. TAD Árbol Binario con su implementación.

Árboles Binarios Ordenados o de Búsqueda

En este tipo de árboles cada nodo tiene asociado un valor, siendo este valor mayor o igual que los valores de los nodos de su subárbol izquierdo y menor o igual que todos los de su subárbol derecho. Según el tipo de aplicación puede no permitirse valores iguales en uno o ambos subárboles. Si se realiza un recorrido en *inorden* por el árbol accederemos a los valores en orden ascendente. Al definir el tipo de datos que representa el valor de un nodo dentro de un árbol binario de búsqueda es necesario que en dicho tipo se pueda establecer una relación de orden. Una ventaja fundamental de los *árboles binarios ordenados*, también llamados *de búsqueda*, es que son en general mucho más rápidos para localizar un elemento que una lista enlazada. Por tanto, son más rápidos para insertar y borrar elementos. Si el árbol está perfectamente equilibrado (esto es, la diferencia entre el número de nodos del subárbol izquierdo y el número de nodos del subárbol derecho es a lo sumo 1, para todos los nodos) entonces el número de comparaciones necesarias para localizar un valor es aproximadamente de logN en el peor caso. Además, el algoritmo de inserción en un árbol binario de búsqueda tiene la ventaja de que no necesita hacer una reubicación de los elementos de la estructura para que esta siga ordenada después de la inserción. El algoritmo de borrado en árboles es algo más complejo, pero más eficiente que el de borrado en un array ordenado. En la figura 4 se representa un ejemplo de un árbol binario ordenado cuyos elementos son números enteros, por lo que resulta muy sencillo distinguir la relación de ordenación entre los diferentes nodos.

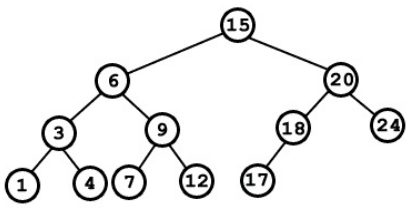

Figura 4. Árbol Binario Ordenado

El TAD Árbol Binario de Búsqueda (*BinarySearchTreeADT*) hereda del TAD Árbol Binario (*BinaryTreeADT*), por lo que en la tabla 2 se especifican las operaciones adicionales que introduce el TAD Árbol Binario de Búsqueda respecto a las ya definidas para el TAD Árbol Binario.

Tabla 2. Operaciones adicionales de un Árbol Binario de Búsqueda.

Operación	Descripción
`addElement`	Añade un elemento al árbol
`removeElement`	Elimina un elemento del árbol
`removeAllOcurrences`	Elimina todas las apariciones de un elemento en el árbol
`removeMin`	Elimina el elemento mínimo del árbol
`removeMax`	Elimina el elemento máximo del árbol
`findMin`	Devuelve una referencia al elemento mínimo del árbol
`findMax`	Devuelve una referencia al elemento máximo del árbol

Los árboles binarios de búsqueda también pueden ser implementados usando una lista enlazada. En concreto para esta práctica usaremos la implementación representada en el figura 5, *LinkedBinarySearchTree* que hereda de su correspondiente implementación para árboles binarios que se detalla en el apartado anterior.

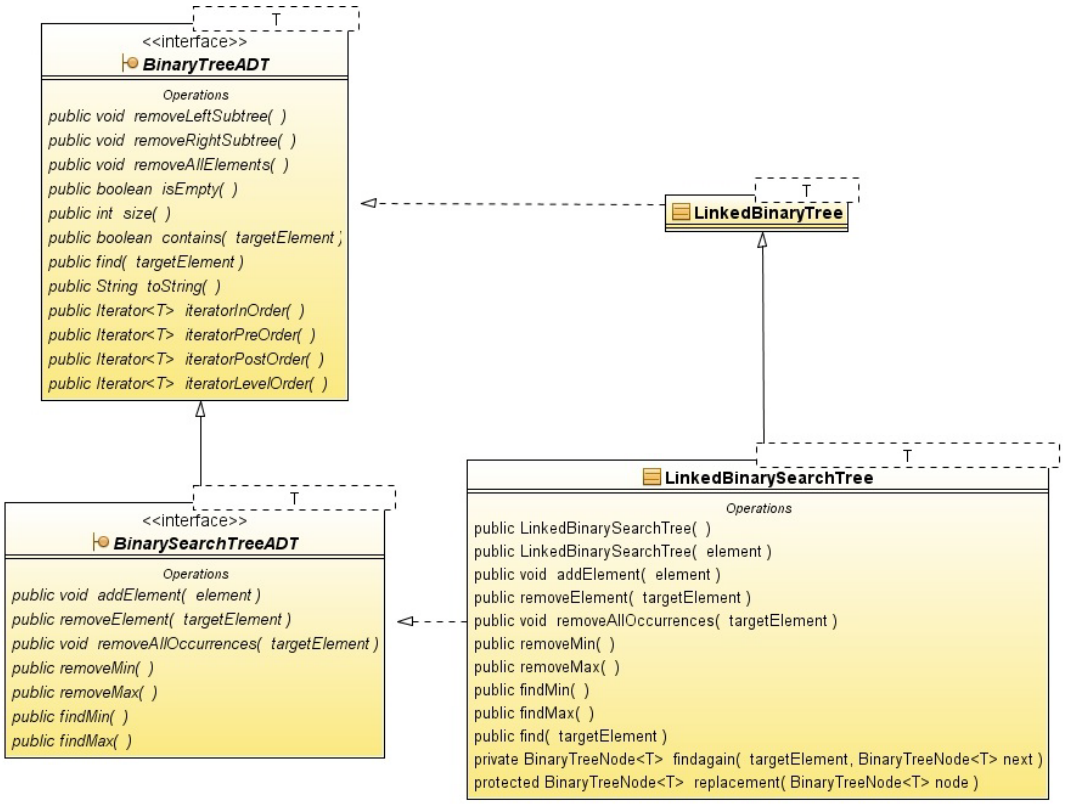

Figura 5. TAD Árboles Binarios de Búsqueda

Preparación de la práctica

Todo el material necesario para resolver los experimentos o problemas se encuentra disponible en WebCT. En la carpeta de esta EPD podrá encontrar:

- Enunciado de esta práctica en PDF.
- Fichero BinaryTrees.jar
- Fichero con los códigos fuentes de árboles binarios: BinaryTrees.zip

Cuando cree un proyecto para resolver un determinado experimento o problema, deberá importar la librería BinaryTrees.jar si necesita trabajar con árboles binarios.

No olvide añadir donde corresponda la cláusula de importación para hacer uso de los TAD y sus implementaciones:

```
import binaryTrees.*;
```

Experimentos

E1. (25 minutos) Cree un proyecto que haciendo uso del TAD Árbol Binario y de su implementación con listas enlazadas construya un árbol de enteros como el de la figura 6. El programa deberá mostrar un menú que permita al usuario decidir si recorre el árbol en preorden, inorden o postorden. A continuación mostrará el recorrido de los nodos del árbol en el orden elegido, el tamaño del árbol, consultará si existe el nodo con valor 5, borrará el subárbol izquierdo, comprobará de nuevo si existe el nodo con valor 5 y mostrará el nuevo tamaño del árbol. Se muestra un ejemplo de ejecución del programa (en cursiva lo que el usuario introduce por teclado):

```
MENÚ
====
[1] Recorrido en Preorden
[2] Recorrido en Inorden
[3] Recorrido en Postorden

Elige una opción: 3
Recorrido Postorden:
5 8 4 9 7 3
Tamaño del árbol: 6
Contiene el elemento 5: true
Borramos subárbol izquierdo
Contiene el elemento 5: false
Nuevo tamaño del árbol: 4
```

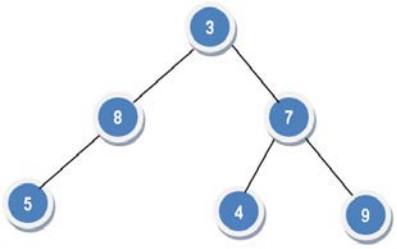

Figura 6. Árbol Binario a representar en el Experimento 1.

E2. (20 minutos) Cree un proyecto que haciendo uso del TAD Árbol Binario de Búsqueda y la implementación con listas enlazadas, construya el árbol binario ordenado de la figura 4. Compruebe qué ocurre al recorrerlo en preorden, inorden y postorden, ¿qué peculiaridad detecta de estos recorridos? Realice llamadas a los distintos métodos que ofrece el TAD para comprobar su funcionamiento.

E3. (15 minutos) Se pretende comprobar el funcionamiento de los árboles binarios ordenados de objetos de la clase *Persona*. Esta clase tiene dos atributos: nombre (*String*) y edad (*int*), con su método constructor, redefinición de *toString* y los métodos consultores y modificadores. Cree un proyecto que permita instanciar un árbol binario de búsqueda de 5 nodos con objetos distintos de esta clase Persona, mostrando su contenido recorriéndolo en Inorden. ¿Funciona correctamente?, ¿se podría solucionar el error estableciendo como orden natural de los objetos de la clase *Persona* el valor del atributo *edad* en orden ascendente?

Ejercicios

EJ1. (45 minutos) El *problema de las concordancias* consiste en que dado un texto contar el número de veces que aparece en él cada palabra, devolviendo un listado alfabético de todas las palabras con su número de ocurrencias. Cree un proyecto que resuelva dicho problema usando el TAD Árbol Binario Ordenado y su implementación con listas enlazadas. El usuario facilitará por teclado las palabras que formarán parte del texto. El programa deberá ignorar la capitalización de las letras que forman las palabras, así la cadena "Algorítmica" será considerada igual a "ALGORÍTMICA" o "algorítmica".

Problemas

P1. (60 minutos). Dado un texto organizado por líneas, el *problema de las referencias cruzadas* consiste en producir un listado de palabras ordenado alfabéticamente, donde cada palabra del texto va acompañada de una lista de referencias que contiene los números de todas las líneas del texto en las que aparece la palabra en cuestión, con posibles repeticiones si la palabra aparece varias veces en una misma línea. Cree un proyecto que resuelva dicho problema ignorando la capitalización de las letras que forman las palabras del texto.

P2. (60 minutos) Descargue de WebCT el fichero *BinaryTrees.zip* con el código fuente de las implementaciones de los árboles binarios. Dentro del paquete *binaryTrees* cree una nueva clase *ExtensionLinkedBinaryTree*, que herede de LinkedBinaryTree, que añada un nuevo método llamado "*refleja"* que dado un árbol binario devuelve otro árbol binario que representa la imagen especular del árbol inicial. Dicha imagen reflejada se consigue cambiando los subárboles izquierdo y derecho en cada nodo excepto la raíz. Cree un programa principal que compruebe el funcionamiento de dicho método.

P3. (60 minutos) Partiendo de la clase *ExtensionLinkedBinaryTree* implementada en el P2, añada otro método llamado *"caminoA"* que tome como parámetro un árbol binario y un ítem, y devuelva una lista con el camino desde la raíz hasta dicho elemento (ambos incluidos). En caso de que el ítem no se encuentre en el árbol, debe devolverse la lista vacía.

P4. (60 minutos) Partiendo de la clase *ExtensionLinkedBinaryTree* implementada en el P2 y P3 añadir nuevos métodos que desarrollen las siguientes operaciones:

- Dado un árbol binario devuelva un natural indicando cuantas ramas posee dicho árbol.
- Dado un árbol binario devuelva un natural indicando la longitud de la rama más larga de dicho árbol.
- Dado un árbol binario devuelva una lista con los elementos que forman la rama más larga de dicho árbol.
- Dado un árbol binario devuelva una lista con los elementos que forman la rama i-ésima de dicho árbol; las ramas se numeran del 1 en adelante y de izquierda a derecha según se dibuja el árbol.

Datos de la Práctica

Autor del documento: David de Vega Rodríguez (marzo 2012).

Revisión del documento:

1. David de Vega Rodríguez (15 marzo 2012)
2. Nombre (Mes año)

Estimación temporal:

- Parte presencial: 120 minutos.
  - Explicación inicial: 15 minutos.
  - Experimentos: 60 minutos.
  - Ejercicio: 45 minutos.
- Parte no presencial: 270 minutos.
  - Lectura y estudio del guión y bibliografía básica: 30 minutos
  - Problemas: 240 minutos