ESTRUCTURAS DE DATOS II

Grado en Ingeniería Informática

CURSO 2020/21

Práctica 2

Árboles

Objetivos

Afianzar los conocimientos adquiridos en teoría en la resolución de problemas usando árboles binarios y árboles binarios de búsqueda

Duración

4 sesiones

Ejercicio 1

Dado un árbol binario, escribir una función que devuelva el número de hojas de dicho árbol.

Ejercicio 2

Diseñar una función que, dado un árbol binario A, devuelva una copia simétrica del mismo. Para el árbol A de la Figura 1, el árbol simétrico que devolvería la función sería el árbol B de la Figura 1.

Figura 1. Árbol binario y su simétrico

Ejercicio 3

Realizar un procedimiento que a partir de un árbol binario, muestre por pantalla los elementos de los nodos visitados mediante un recorrido en zigzag, comenzando por un sentido dado ('I' para subárbol izquierdo, 'D' para subárbol derecho):

```
template <typename T>
void recorridoZigzag( const Arbin<T>& a, char sentido )
```

Ejercicio 4

Un árbol binario está *compensado* si, para cada nodo interno del árbol, la diferencia entre el número de nodos de sus hijos es menor o igual que 1. Escribir una función que devuelva verdadero si un árbol está compensado y falso en caso contrario. Se entiende que el árbol vacío y las hojas están compensados.

Ejercicio 5

Cada nodo de un árbol binario A almacena una letra. La concatenación de las mismas, en cada camino que va desde la raíz a una hoja representa una palabra. Realizar una función que visualice todas las palabras almacenadas en un árbol binario A.

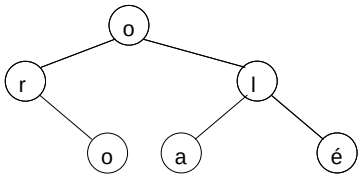

Figura 2. Árbol binario de letras

Por ejemplo, para el árbol de la Figura 2 deberían visualizarse las siguientes palabras: oro, ola y olé.

Ejercicio 6

Sea *a* un árbol de búsqueda binario de enteros positivos y *x* un número entero positivo. Se pide escribir una función que dados *a* y *x* devuelva el siguiente elemento mayor que *x* almacenado en el árbol. En caso de que *x* no tenga un elemento mayor en el árbol *a* debe lanzar una excepción. El elemento *x* puede estar en el árbol o no.

```
int siguienteMayor(const ABB<int>& a, int x)
     throw (NoHaySiguienteMayor)
```

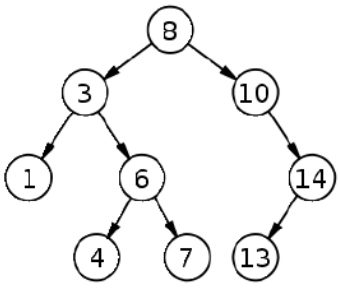

Figura 3. Árbol binario de bíusqueda de enteros positivos

En el árbol de la Figura 3, la llamada *siguienteMayor(a,* 5*)* debe devolver **6**. Sin embargo, para la llamada *siguienteMayor(a,* 14*)* debe lanzar la excepción *noHaySiguienteMayor*.

Ejercicio 7

Realizar una función que devuelva la posición de un elemento en un ABB (desde 1 hasta el número de nodos del árbol) según un recorrido en inorden de dicho árbol, pero sin tener que procesar a posteriori el resultado de dicho recorrido. Si el elemento no está en el árbol, la función devolverá 0.

Ejercicio 8

Sea *a* un árbol binario de enteros. Denominamos *suma de un camino* a la suma de los valores de los nodos que forman un camino que vaya desde la raíz del árbol a una de sus hojas. Se pide escribir una función booleana que dado *a* y un valor entero devuelva cierto si dicho valor coincide con alguna suma de camino en dicho árbol y falso si no coincide con ninguna. Para un árbol vacío la función *haySumaCamino* devolverá cierto para un valor de *suma* 0 y falso en cualquier otro caso.

```
bool haySumaCamino(const Arbin<int>& a, int suma)
```

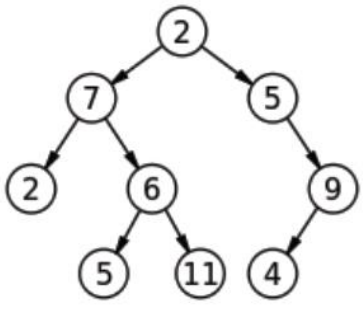

Figura 4. Árbol binario de enteros

En el árbol de la Figura 4, la llamada *haySumaCamino(a,* 26*)* debe devolver **verdadero** ya que el camino 2 – 7 – 6 – 11 suma 26. Sin embargo, para la llamada *haySumaCamino(a,* 9*)* debe devolver **falso**, ya que no hay ningún camino que vaya desde la raíz a una hoja que sume 9.