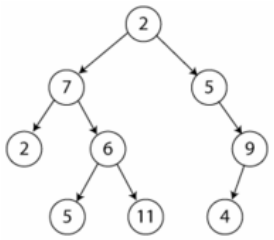
Práctica 4. Implementación de árboles y algoritmos de recorrido

Ejercicio 0: Representa mediante la implementación de árbol binario con nodos enlazados (la vista en clase) el siguiente árbol. Después imprímelo haciendo un recorrido sobre los nodos del mismo.



Ejercicio 1: Diseñar una función que lea un fichero de texto con la estructura de un árbol como tabla de contenidos y cree el árbol correspondiente (supondremos un árbol binario). El texto del fichero tendrá un aspecto equivalente al siguiente:

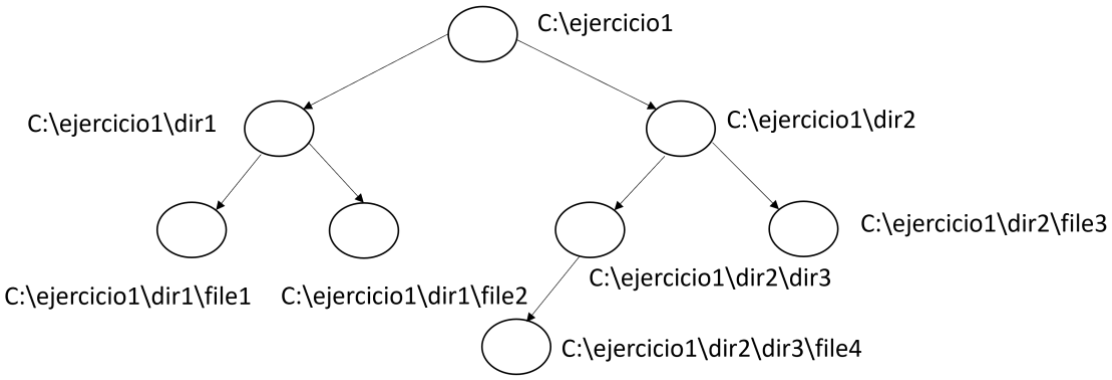
```
Título documento
      1. Introducción
            1.1.  Motivación
                  1.1.1.  Cosas iniciales
                  1.1.2.  Cosas finales
                        1.1.2.1. Primera cosa
                        1.1.2.2. Segunda cosa
            1.2.  Contribución
      2. Estado del arte
            2.1. Referencias globales
            2.2. Referencias cercanas
```

La indentación del texto (mediante un carácter de tabulador) será lo que determinará el anidamiento de los nodos del árbol. Para comprobar el correcto funcionamiento de la nueva función, implementa un código de prueba que haga uso de un árbol guardado en un fichero de texto.

Ejercicio 2: Implementar una función equivalente a la del Ejercicio 1, pero en este caso el elemento de cada nodo del árbol contendrá más de un atributo y en el fichero de texto aparecerá como: *atr1,atr2,atr3* donde la coma delimita cada atributo.

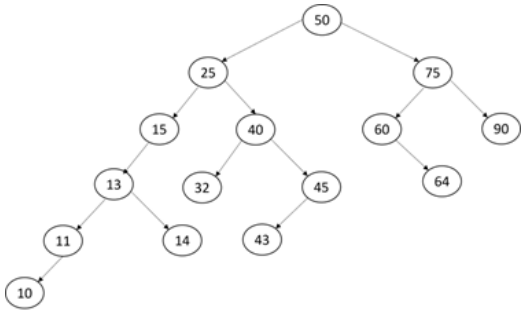
Ejercicio 3: Modifica la implementación de LinkedBinaryTree para que incorpore el método especial `__str__` que permita imprimir un árbol binario, según un algoritmo de recorrido acorde con el resultado deseado. Comprueba el correcto funcionamiento para comprobar que la impresión del árbol generado en el Ejercicio 1 es igual (salvo la indentación) que el texto del fichero de entrada. Modifica posteriormente la implementación del árbol para que la cadena que dé como salida el método `__str__` muestre los títulos por niveles: título principal, todos los títulos de segundo nivel, todos los títulos de tercer nivel, etc.

Ejercicio 4: Construir un árbol binario que represente una estructura de directorios que exista en la máquina local. A continuación, se incluye un ejemplo:



El árbol se leerá desde un fichero de texto usando la función del Ejercicio 1. Recorrer el árbol anterior para calcular el espacio en disco usado por el directorio. Usar el método `getsize` del módulo `os` para obtener el tamaño en disco de una ruta especificada.

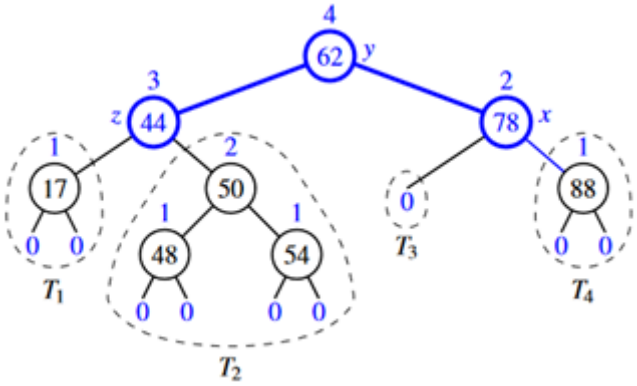
Ejercicio 5: Desarrollar una función que, recibiendo como argumento un árbol binario, devuelva como resultado la suma de las claves impares contenidas en nodos con un hijo. Por ejemplo, dado el árbol siguiente, los nodos que cumplen estas condiciones son 45, 11 y 15, luego el resultado será 71.



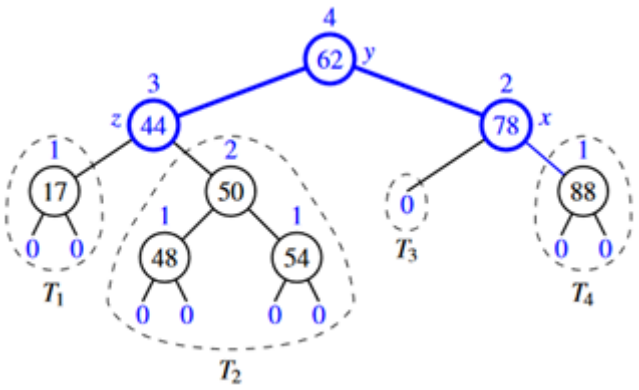
Ejercicio 6: Si insertamos las entradas (1,A), (2,B), (3,C), (4,D), y (5,E), en este orden, en un árbol de búsqueda binaria vacío, ¿qué estructura tendría el árbol?

Ejercicio 7: ¿Cuántos árboles de búsqueda binaria diferentes pueden almacenar las claves 1, 2, 3?

Ejercicio 8: Dibujar el árbol AVL resultante de insertar una entrada con clave 52 en el árbol AVL siguiente:



Ejercicio 9: Dibujar el árbol AVL resultante de borrar la entrada con clave 62 del árbol siguiente:



Ejercicio 10: Desarrollar una función que dado un árbol binario T, devuelva una copia idéntica del mismo.

Ejercicio 11: Desarrollar una función que visualice los nodos que están en el nivel de profundidad *n* de un árbol binario.

Ejercicio 12: Escribe una función que devuelva el número de hojas de un árbol binario.

Ejercicio 13: Desarrollar una función que indique si un árbol es un árbol de búsqueda binaria.

Ejercicio 14: Desarrollar una función que indique si un árbol binario es completo.

Ejercicio 15: Supongamos que cada nodo de un árbol binario almacena una letra. La concatenación de las mismas, en cada camino que va de la raíz a una hoja representa una palabra. Desarrollar una función que visualice todas las palabras almacenadas en el árbol binario.

Ejercicio 16: Haz uso de la implementación vista para árbol de búsqueda binaria para realizar las siguientes tareas:

1. Crea una lista con una secuencia numérica desde 0 a 100.000 y "baraja" su contenido para desordenarla con el método `shuffle` del módulo `random`.
2. Inserta los números en un árbol binario de búsqueda, usando como clave y valor el propio número.
3. Realiza una búsqueda de cada uno de los números en la lista original, y luego haz lo mismo, pero usando el árbol binario de búsqueda, comparando el tiempo que emplean las dos tareas.
4. Realiza la misma operación que en el punto anterior, pero eliminando los números.
5. Realiza variaciones en la cantidad de números utilizados y valida los órdenes de complejidad vistos en teoría.