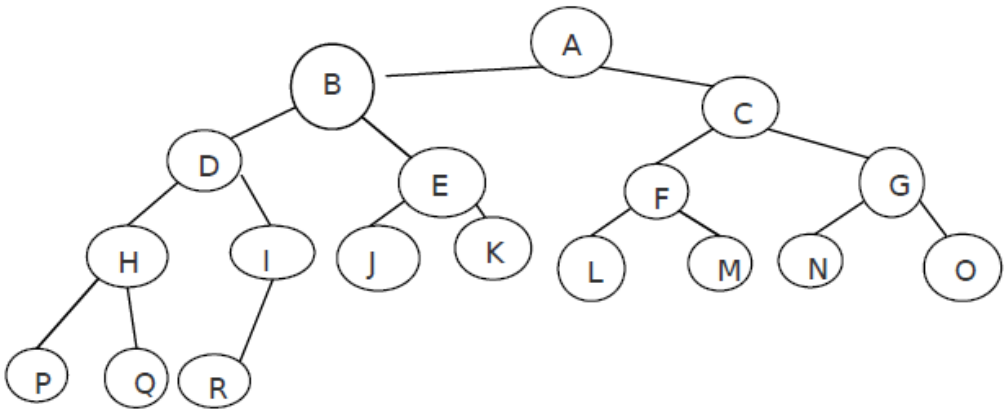
PROGRAMACIÓN 2 - Unidad 5: Árboles
Ejercicios

1. Considérese el árbol siguiente



   a. ¿Cuál es su profundidad?
   b. ¿Es un árbol binario? ¿Es completo o casi-completo? Justifique la respuesta.
   c. ¿Cuál es el **padre** del nodo R?
   d. ¿Cuáles son los **antecesores** del nodo J?
   e. Recorra el árbol anterior según los algoritmos de preorden, orden medio, postorden y en anchura. Para cada uno de los algoritmos dibuja el árbol e indica en cada nodo el orden de visita; y muestra al final los nombres de los nodos según el orden de visita.

2. Escriba el pseudocódigo de los algoritmos apropiados para determinar:
   a. La cantidad de nodos de un árbol binario
   b. El número de hojas de un árbol binario.
   c. La suma del contenido de los nodos de un árbol binario de elementos de tipo entero.
   d. La profundidad de un árbol binario.

   Para los cuatro problemas, debe pensar cómo depende el valor a calcular del propio nodo y del valor correspondiente a los subárboles que lo tienen como raíz.

3. Escriba el código C para implementar los pseudocódigos de los algoritmos del problema anterior. Para ello, puedes asumir las siguientes estructuras de datos y definiciones:

```
typedef enum {FALSE = 0, TRUE=1} Bool;
typedef enum {ERROR = 0, OK=1} Status;

#define info(pnode) ((pnode)->info)
#define left(pnode) ((pnode)->left)
#define right(pnode) ((pnode)->right)
#define root(pt) ((pt)->root)

struct _BTNode {
      Element *info;
      struct _BTNode *left;
      struct _BTNode *right;
};
typedef struct _BTNode BTNode;
```

```
struct _BinaryTree {
        BTNode *root; // un árbol es el puntero a su nodo raíz
};
typedef struct _BinaryTree BinaryTree;
```

4. Escriba el pseudocódigo de un algoritmo para determinar si un árbol binario es un árbol binario de búsqueda.

5. Escriba el pseudocódigo de un algoritmo para determinar si un árbol binario es:
   a. Estrictamente binario (es decir, todo nodo no-terminal tiene **exactamente** dos hijos)
   b. Completo.

6. Escriba el código C, sin control de errores, de una función para determinar si un árbol binario es estrictamente cuasi-completo (es decir, cuasi-completo, y además todos los nodos del último nivel ocurren a la izquierda). Puede utilizar los tipos y estructuras de datos y macros en C del ejercicio 3. Sugerencia: Utilice una variación de la búsqueda en anchura.

7. a) Escribir una función en C que reciba un árbol binario (pero no necesariamente un ABdD) y un elemento y devuelva el nivel del árbol en que se encuentra, o -1 si no se encuentra en el árbol. Puedes utilizar las estructuras de datos y macros del ejercicio 3.
   b) Si el árbol tuviese nodos repetidos debería devolver el nodo más profundo.

   Suponed que se dispone de una función que permite comparar dos elementos con el prototipo:
   `int element_compare (Element *ele1, Element  *ele2)`
   tal como se ha descrito en la presentación del tema para ABdDs. (Es decir, devuelve 0 si ele1 == ele2, un número negativo si ele1 < ele2, y un número positivo si ele1 > ele2).

8. Repetir el problema anterior suponiendo que el árbol es un árbol binario de búsqueda (ABdB). ¿El algoritmo del ejercicio 7 sería eficiente para un ABdB? ¿Por qué?

9. Proporcione el código C de una función de prototipo
   `void tree_printFromLevel (BinaryTree *pa, int level, FILE *pf)`
   que imprima en un flujo de salida pf el campo info de todos los nodos del árbol binario completo `pa` que estén en un nivel menor o igual al nivel `level` que se le pasa como argumento. Si la profundidad del árbol fuese inferior al nivel, la función devolverá ERROR y no imprimirá nada. Puedes utilizar las estructuras de datos y macros en C del ejercicio 3.

10. Escriba el código en C para recorrer un árbol binario en orden previo y en orden posterior. Puedes utilizar las estructuras de datos y macros en C del ejercicio 3, y el pseudocódigo de la presentación del tema.

11. Dibujar los árboles de expresión que representan las siguientes expresiones
    (A + B) / (C - D);
    A + B + C / D;
    A – (B - (C - D) / (E + F));
    (A + B) * ((C + D) / (E + F));

12. Construir algorítmicamente el árbol de expresión a partir de las siguientes expresiones sufijo:
    A B C * D - E F / G / - * ;
    A B + C D - / E F * G - - ;
    A B C D - - E F G + + * * ;

13. Utilizando los árboles obtenidos en el problema anterior, obtenga las expresiones prefijo e infijo correspondientes a las expresiones sufijo.

14. Supóngase que al recorrer un árbol binario en preorden se obtiene la siguiente lista
    G B Q A C K F P D E R H
    Dibuje una posible estructura de nodos del árbol. ¿Hay otras estructuras posibles?

15. Construir el árbol binario de búsqueda a partir de cada una de las listas siguientes:
    G B Q A C K F P D E R
    A B C D E F G H
    15 22 12 35 31 13 10 27 6 9 25

16. En un árbol binario de búsqueda se define el predecesor de un dato D como el mayor dato D' del árbol tal que D' < D y su sucesor como el menor dato del árbol tal que D' > D. Sobre los arboles del problema anterior, encontrar el predecesor y el sucesor de cada uno de los elementos.

17. Escribir el código de C de un algoritmo que devuelva un puntero al elemento mínimo de un árbol binario de búsqueda. Dar un procedimiento recursivo y otro no recursivo.
    ```
    Element *MinElement (BinaryTree *pa)
    ```

18. Dar el pseudocódigo y el código C de una función que reciba un nodo de un árbol binario de búsqueda y devuelva el campo info de su nodo sucesor. Para ello, considere que la estructura de datos utilizada para representar los nodos del árbol contiene un campo adicional `parent` que permite acceder al **padre** de un nodo dado.

19. Dos árboles binarios son topológicamente similares si ambos están vacíos o, si ambos no están vacíos, sus subárboles izquierdos son similares y sus subárboles derechos son similares. De un ejemplo de árboles similares de profundidad mayor o igual a dos.
    a. Escriba el pseudocódigo de un algoritmo para determinar si dos árboles son similares.
    b. Utilizando las estructuras de datos y macros en C del ejercicio 3 proporcione el código C de una función con el prototipo siguiente que implemente el algoritmo anterior.
    ```
    Bool similarTrees (BinaryTree *T1, BinaryTree *T2)
    ```

20. Escriba el pseudocódigo de un algoritmo que reciba un árbol binario y cree un nuevo árbol binario copia del anterior. Dar, a continuación, el código C del algoritmo propuesto. Suponga, para ello, los tipos y estructuras de datos definidas en el ejercicio 3.

21. Escriba el pseudocódigo de un algoritmo que acepte un árbol binario y lo modifique de forma que sea la imagen refleja del original (es decir, un árbol en el que todos los subárboles derechos del árbol original son ahora subárboles izquierdos y viceversa). Dar, a continuación, el código C del algoritmo propuesto.

22. Escriba el pseudocódigo y el código C de un algoritmo recursivo que devuelva como cadena de caracteres la representación prefijo de una expresión aritmética almacenada en un árbol de expresión.

23. Se desea construir un algoritmo que, dado un árbol binario de búsqueda (ABdB), un elemento y un flujo de salida, busque dicho elemento en el árbol y si se encuentra imprima en el flujo de salida el camino desde el nodo raíz del árbol hasta el elemento. Es importante que sólo se imprima el camino si se encuentra el nodo en el árbol.

    Puede usar las funciones de la interfaz de cualquiera de los TADs vistos durante el curso, así como las estructuras y definiciones del ejercicio 3 y las primitivas que necesites para gestionar el contenido del campo info (element_print, element_free, etc.)

    El algoritmo sólo se considerará válido si tiene la menor complejidad posible.

    a) Proporcione el código C con control de errores de la función de prototipo *int tree_printPath (FILE *pf, AB *T, Element *ele)* que implementa el algoritmo anterior.

    b) ¿Cuál es la complejidad de la función desarrollada? Responda justificadamente.