Relación de ejercicios sobre árboles y tablas hash

Esta entrega corresponde a la relación de ejercicios sobre átboles y tablas hash.

Debajo de cada ejercicio puedes encontrar la función que debes implementar. Además, debe haber una función main con las distintas pruebas que has realizado para verificar el funcionamiento de la función.

Si un ejercicio no funciona correctamente o se tiene claro que no resuelve todos los casos posibles, se debe explicar adecuadamente.

Los ficheros no deben dar errores de compilación.

Cualquier entrega que no cumpla con estas normas no será tenida en cuenta.

Ejercicios:

1. Escribe una función que determine cual es la hoja más profunda de un árbol binario.
   `typename bintree<T>::node mas_profunda(const bintree<T> &Arb)`
2. Escribe una función que acepte un objeto de la clase *node* y un árbol general *T* y devuelva el nivel del nodo en el árbol.
   `int nivel(const bintree<T> &Arb, typename bintree<T>::node n)`
3. Construye una función que determine si un árbol binario es completo.
   `bool completo(const bintree<T> &Arb)`
4. Escribe una función que realice la reflexión de un árbol binario. Es decir, que intercambie los hijos de cada uno de los nodos del árbol.
   `void relexion(bintree<T> &Arb)`
5. Definimos densidad de un árbol binario A como la suma de las profundidades de las hojas de un árbol. Escribe una función para calcular la densidad de un árbol binario.
   `int densidad(const bintree<T> &Arb)`
6. Dado un árbol de expresión representado mediante el tipo bintree en el que tenemos un carácter indicando el nombre de una variable o un operador *, +, -, /. Escribe una función que devuelva el resultado de la evaluación de la expresión guardada en el árbol.
   `int evalua(const bintree<char> &Arb, const map<char, int> &valores)`
7. Implementa una función que dada una expresión en postfijo devuelva el árbol binario asociado. Los operandos serán letras y los operadores +,-,*,/
   `bintree<char> construir_arbol(string expresion)`
8. Escribe una función no recursiva (usando una pila) para calcular la altura de un árbol binario.
   `int altura(const bintree<T> &Arb)`
9. Dos árboles binarios son similares si son iguales en cuanto a su estructura, es decir, cada nodo en un árbol tiene los mismos hijos y en el mismo lugar que el correspondiente en el otro árbol (sin importar

las etiquetas). Escribe una función que dados dos árboles binarios devuelva si son o no similares.
`bool similares(const bintree<T> &Arb1,const bintree<T> &Arb2)`

10. Dado un bintree Arb, organizado como un BST, implementa una función a la que se le pasen dos valores a y b y que determine de forma eficiente los valores presentes en el árbol que estén comprendidos entre ambos. Tanto a como b no tienen porqué aparecer en el árbol.
`list<T> comprendidos(const bintree<T> &Arb, T a, T b)`

11. El recorrido en preorden de un determinado árbol binario es:

GEAIBMCLDFKJH

y en inorden

IABEGLDCFMKHJ

Resolver:

a) Dibuja el árbol binario.

b) Da el recorrido en postorden.

c) Diseña una función para dar el recorrido en postorden dados los recorridos en preorden e inorden (sin construir el árbol) y escribe un programa para comprobar el resultado del apartado anterior.
`list<T> postorden(const list<T> &preorden, const list<T> &inorden)`

12. Construye un BST y un POT con las claves 50,25,75,10,40,60,90,35,45,70,42.

13. Indica la secuencia de rotaciones resultante de la inserción del conjunto de elementos {1, 2, 3, 4, 5, 6, 7, 15, 14, 13, 12, 11, 10, 9, 8} en un árbol AVL.

14. ¿Bajo qué condiciones puede un árbol ser parcialmente ordenado y binario de búsqueda simultáneamente? Razona la respuesta.

15. Implementa una función que devuelva el nodo que sea el primer ancestro común a los nodos $n1$ y $n2$. La eficiencia debe ser proporcional a la altura del árbol.

```
typename bintree<T>::node ancestro_comun(typename bintree<T>::node n1,
typename bintree<T>::node n2)
```

16. Dada una secuencia de claves $S = \{5, 8, 4, 13, 66, 2, 9, 12, 11, 17\}$

- Inserta la secuencia anterior, en el orden indicado, en una tabla hash cerrada con resolución de colisiones lineal y que tiene tamaño 11. A continuación, borrar los elementos 2 y 17.
- Construye el árbol AVL que se obtiene al insertar los elementos de la secuencia S (en el orden en que aparecen).

17. Compara empíricamente la eficiencia de insertar miles de claves aleatorias en un set frente a hacerlo en un unordered_set. Redacta un pequeño informe sobre la comparación.

18. Dada la representación `vector<list<T> > TH` de un tipo de dato `hash_table<T>`. Diseña una representación del iterador e implementa las operaciones:

- `hash_table<T>::iterator::operator--(),tabla_hash<T>::begin()` y
- `hash_table<T>::end()`.