Capítulo 6. ÁRBOLES.

6.1 Árboles binarios.

Un árbol binario es un conjunto finito de elementos, el cual está vacío o dividido en tres subconjuntos separados:

- El primer subconjunto contiene un elemento único llamado **raíz del árbol**.
- El segundo subconjunto es en sí mismo un árbol binario y se le conoce como **subárbol izquierdo** del árbol original.

- El tercer subconjunto es también un árbol binario y se le conoce como **subárbol derecho** del árbol original.

El subárbol izquierdo o derecho puede o no estar vacío.

Cada elemento de un árbol binario se conoce como **nodo** del árbol.

La Ilustración 2 muestra una representación de un árbol binario.

Ejercicio: ¿Una lista podría ser un árbol binario? ¿Una lista doblemente enlazada? ¿Por qué? ¿Qué otros ejemplos podrían o no considerarse como árboles binarios?

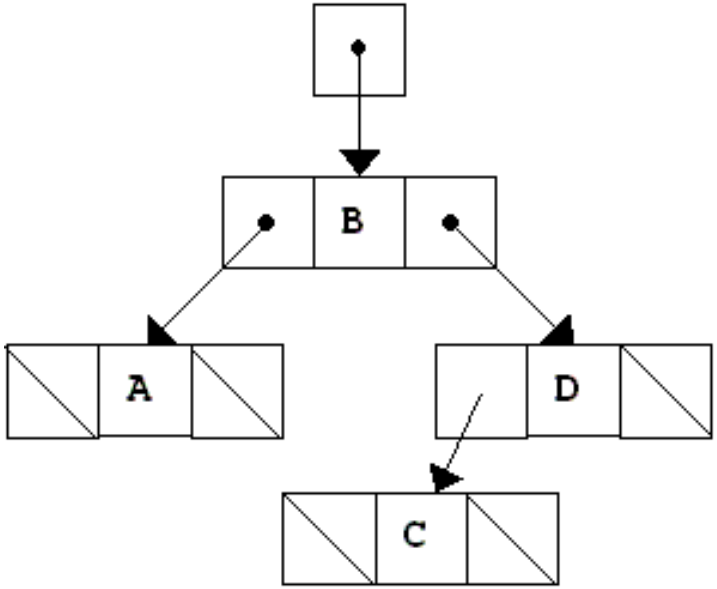

Ilustración 2 Árbol binario.

Si *B* es la raíz de un árbol binario y *D* es la raíz del subárbol izquierdo/derecho, se dice que *B* es el **padre** de *D* y que *D* es el **hijo izquierdo/derecho** de *B*.

A un nodo que no tiene hijos, tal como *A* o *C* de la Ilustración 2, se le conoce como **hoja**.

Un nodo *n1* es un **ancestro** de un nodo *n2* (y *n2* es un **descendiente** de *n1*) si *n1* es el padre de *n2* o el padre de algún ancestro de *n2*.

Recorrer un árbol de la raíz hacia las hojas se denomina **descender** el árbol y al sentido opuesto **ascender** el árbol.

Un **árbol estrictamente binario** es aquel en el que cada nodo que no es hoja, tiene subárboles izquierdo y derecho que no están vacíos.

Un árbol estrictamente binario con *n* hojas siempre contiene *2n-1* nodos.

El **nivel** de un nodo en un árbol binario se define del modo siguiente:

1. La raíz del árbol tiene el nivel 0.
2. El nivel de cualquier otro nodo en el árbol es uno más que el nivel de su padre.

La **profundidad o altura** de un árbol binario es el máximo nivel de cualquier hoja en el árbol.

Un **árbol binario completo** de profundidad *p*, es un árbol estrictamente binario que tiene todas sus hojas en el nivel *p*.

6.2 Operaciones en árboles binarios.

Se aplican varias operaciones primitivas a un árbol binario.

Si *p* es un apuntador a un nodo *nd* de un árbol binario:

1. La función *info(p)* regresa el contenido de *nd*.
2. La función *left(p)* regresa un apuntador al hijo izquierdo de *nd*.
3. La función *right(p)* regresa un apuntador al hijo derecho de *nd*.

4. La función *father(p)* regresa un apuntador al padre de *nd*.

5. La función *brother(p)* regresa un apuntador al hermano de *nd*.

6. La función *isLeft(p)* regresa *true* si *nd* es un hijo izquierdo de algún otro nodo en el árbol, y *false* en caso contrario.

7. La función *isRight(p)* regresa *true* si *nd* es un hijo derecho de algún otro nodo en el árbol, y *false* en caso contrario.

En la construcción de un árbol binario son útiles las operaciones:

1. *makeTree(x)* crea un nuevo árbol que consta de un nodo único con un campo de información *x*, y regresa un apuntador a este nodo.
2. *setLeft(p, x)* crea un nuevo hijo izquierdo de *node(p)* con el campo de información *x*.
3. *setRight(p, x)* crea un nuevo hijo derecho de *node(p)* con el campo de información *x*.

6.3 Aplicaciones de árboles binarios.

Un árbol binario es una estructura de datos útil cuando deben tomarse decisiones en dos sentidos en cada punto de un proceso.

Suponga que se desea encontrar todos los duplicados de una lista de números.

Considérese lo siguiente:

1. El primer número de la lista se coloca en un nodo que se ha establecido como la raíz de un árbol binario con subárboles izquierdo y derecho vacíos.
2. Cada número sucesivo en la lista se compara con el número en la raíz, aquí se tienen 3 casos:
    a. Si coincide, se tiene un duplicado.
    b. Si es menor, se examina el subárbol izquierdo.

c. Si es mayor, se examina el subárbol derecho.

3. Si alguno de los subárboles esta vacío, el número no es un duplicado y se coloca en un nodo nuevo en dicha posición del árbol.

4. Si el subárbol no está vacío, se compara el número con la raíz del subárbol y se repite todo el proceso con el subárbol.

Un **árbol binario de búsqueda (ABB)** no tiene valores duplicados en los nodos y además, tiene la característica de que:

1. Los valores en cualquier subárbol izquierdo son menores que el valor en su nodo padre.

2.Los valores en cualquier subárbol derecho son mayores que el valor en su nodo padre.

El árbol binario de búsqueda de la Ilustración 3 fue construido dada la siguiente secuencia de elementos:

14, 15, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17

Ejercicio: realice la inserción de los elementos anteriores y compare su árbol con el de la Ilustración 3.

Una operación común es **recorrer** todo un árbol binario en un orden específico.

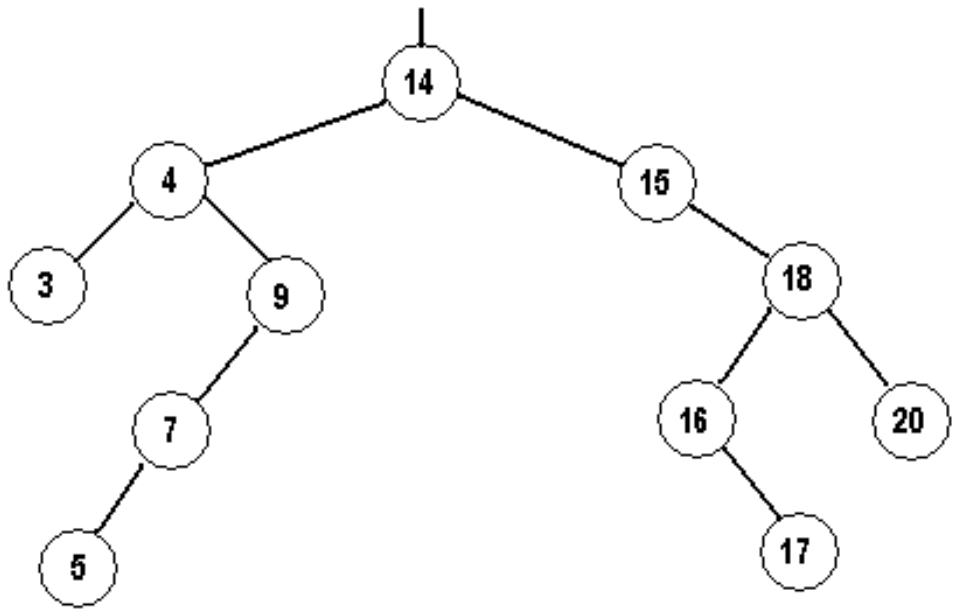

Ilustración 3 Árbol binario de búsqueda.

A la operación de recorrer un árbol de una forma específica y de "numerar" sus nodos, se le conoce como **visitar** el árbol (procesar el valor del nodo).

Ejercicio: ¿Cuál es el orden natural de recorrido de una lista? ¿Cuál sería el orden natural de recorrido de un árbol?

En general, se definen tres métodos de recorrido de un árbol binario.

Antes de presentarlos se deberán tener en mente las siguientes consideraciones:

1. No se necesita hacer nada para un árbol binario vacío.
2. Todos los métodos se definen recursivamente.
3. Siempre se recorren la raíz y los subárboles, la diferencia radica en el orden en que se visitan.

Para recorrer un árbol binario no vacío en **orden previo** (**orden de primera profundidad**) se ejecutan tres operaciones:

1.Visitar la raíz.

2.Recorrer el subárbol izquierdo en orden previo.

3.Recorrer el subárbol derecho en orden previo.

Para recorrer un árbol binario no vacío en **orden** (**orden simétrico**) se ejecutan tres operaciones:

1.Recorrer el subárbol izquierdo en orden.

2.Visitar la raíz.

3.Recorrer el subárbol derecho en orden.

Para recorrer un árbol binario no vacío en **orden posterior** se ejecutan tres operaciones:

1.Recorrer el subárbol izquierdo en orden posterior.

2.Recorrer el subárbol derecho en orden posterior.

3.Visitar la raíz.

Ejercicios a desarrollar con árboles:

1.Ordenamiento de números e identificación de elementos repetidos almacenados en una lista.

2.Árboles de expresiones.

3.Determinar el número de nodos de un árbol binario.

4.La suma de todos los nodos.

5.La profundidad de un árbol binario.

6.Determinar si un árbol binario es o no estrictamente binario.

7.Determinar si un árbol binario es o no completo de nivel *p*.

6.4 Eliminación de un ABB.

La eliminación es el problema inverso a la inserción, sin embargo, las cosas no son tan sencillas como para la inserción.

Si el nodo que se pretende eliminar es un nodo hoja o un nodo con un solo descendiente, la eliminación es directa.

La dificultad radica en la eliminación de un nodo con dos descendientes. En este caso, el elemento eliminado será substituido por el descendiente más a la derecha de su subárbol izquierdo (o bien por el descendiente más a la izquierda de su subárbol derecho).

Obsérvese que estos nodos substitutos tienen a lo más, un descendiente.

Lo anterior queda mejor representado en la Ilustración 4.

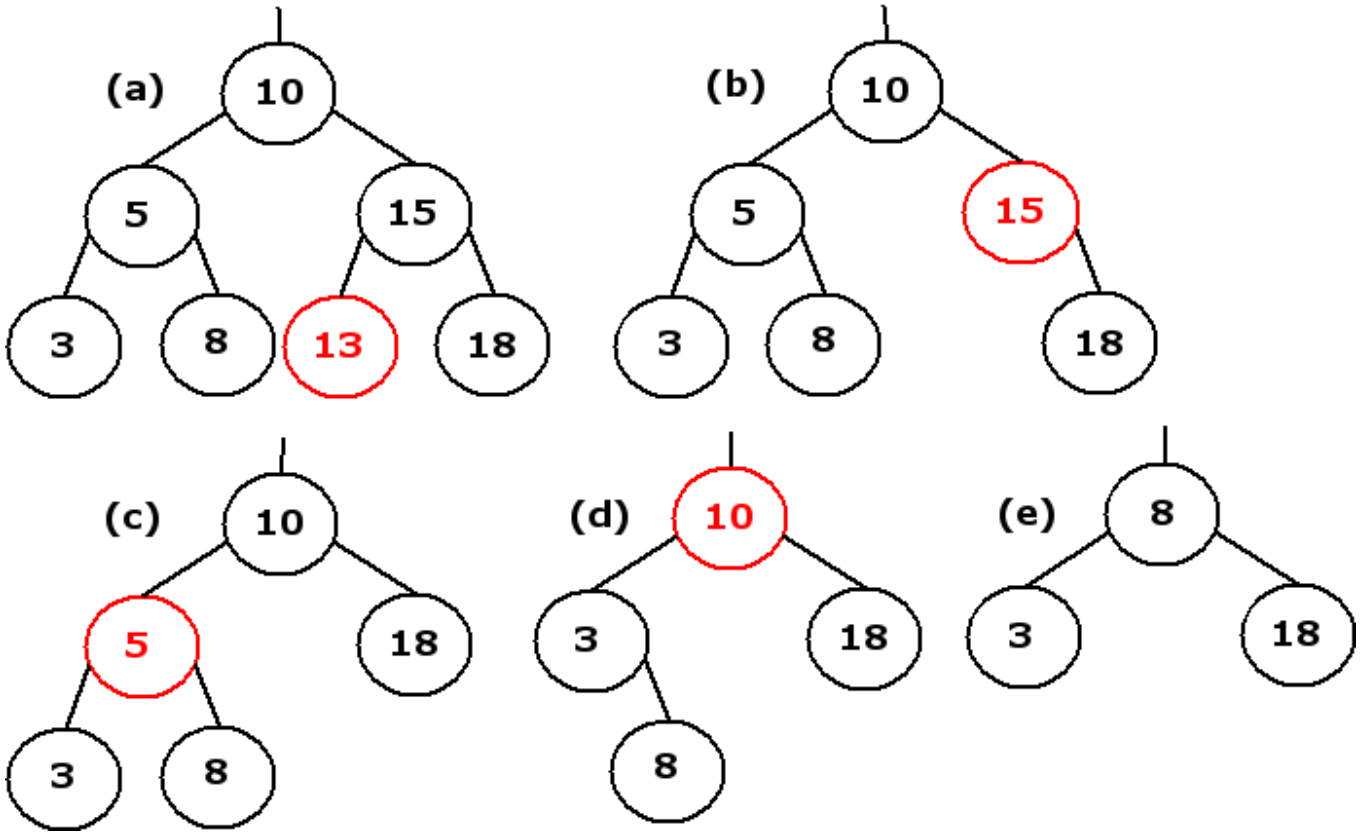

Ilustración 4. Eliminación de un ABB.

Ejercicio: En base al proceso descrito con anterioridad, inserte en un ABB los números: **9, 18, 10, 2, 6, 21, 8, 1, 7**. Después elimine los números **21, 18 y 9**.

Ejercicio: Realice la implementación de la eliminación de nodos en un ABB, considere los tres casos previstos:

1. Eliminación de nodos hojas.
2. Eliminación de nodos con un solo hijo.
3. Eliminación de nodos con dos hijos (implemente el esquema de substitución del hijo más a la derecha del subárbol izquierdo).