

Introducción



- 1) Hasta el momento se han visto estructuras de datos lineales, a cada elemento siempre le precede como máximo otro elemento.
- 2) Pero la estructura de datos árboles introducen el concepto de ramificación entre componentes (a un elemento le puede preceder o suceder varios elementos).
- 3) Los árboles son las estructuras de datos no lineales y dinámicas más importantes en el área de la computación.
 - 1) Dinámicas porque pueden cambiar tanto de forma como tamaño durante la ejecución del programa.
 - 2) No lineales porque cada elemento en el árbol puede tener más de un sucesor.

Árboles



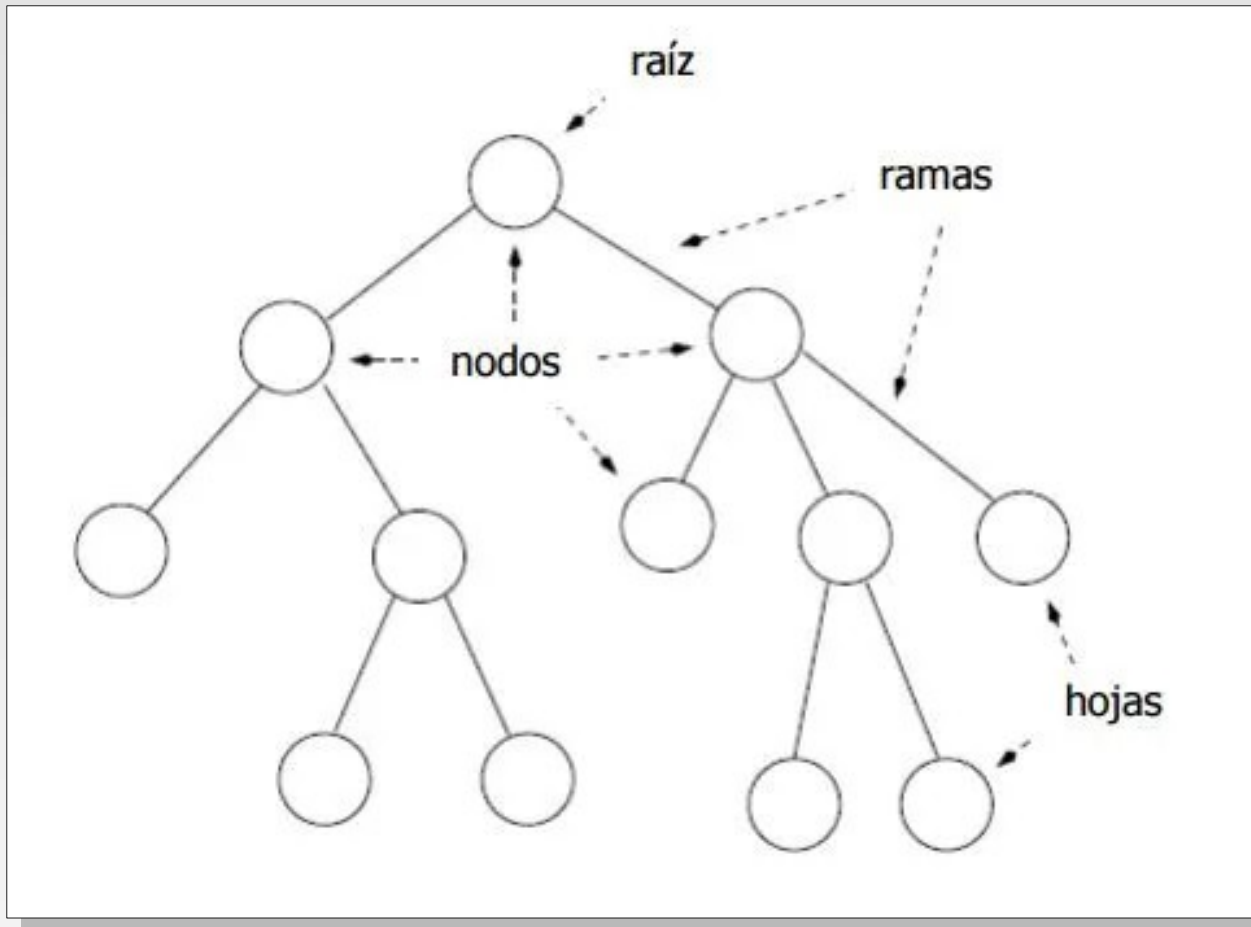
- 1) Los árboles son una estructura de datos jerárquica que se utilizan en informática para almacenar y organizar información.
- 2) Un árbol impone una estructura jerárquica sobre una colección de objetos.
- 3) Los árboles se utilizan comúnmente para:
 - 1) Analizar circuitos eléctricos
 - 2) Representar la estructura de fórmulas matemáticas
 - 3) En algoritmos de búsqueda, ordenamiento y recuperación de información en sistemas de bases de datos.
 - 4) Para representar la estructura sintáctica de un programa fuente en los compiladores.

Terminología fundamental

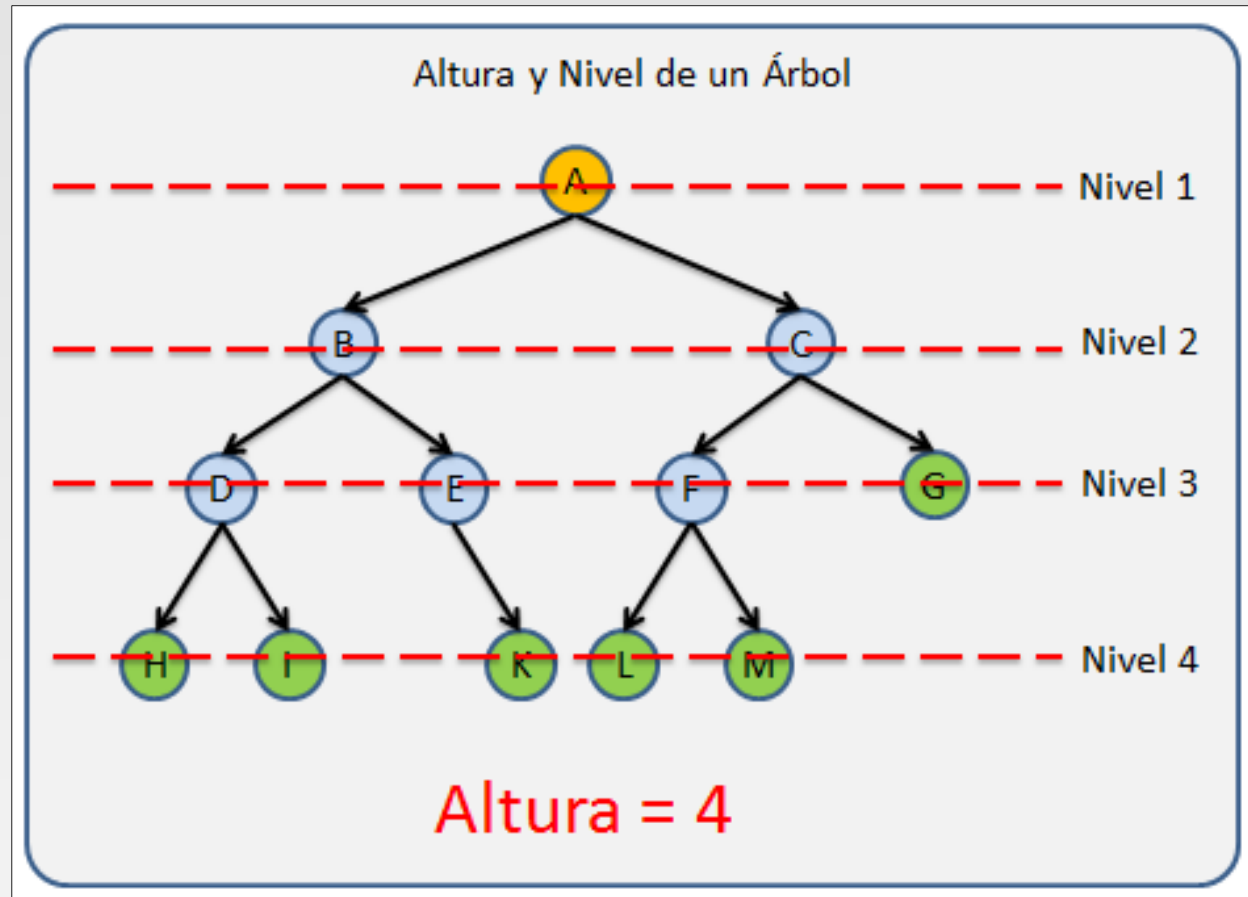


- 1) Un árbol es una colección de elementos llamados **nodos**.
- 2) El **nodo raíz** es el nodo principal del árbol y todos los demás nodos están conectados a él.
- 3) Un árbol es un conjunto de nodos conectados por enlaces, donde cada nodo (excepto el nodo raíz) tiene un único padre y puede tener varios hijos.
- 4) A menudo se representa un nodo por medio de una letra, una cadena de caracteres o un círculo con un número en su interior.

Terminología fundamental



Terminología fundamental



Tipos de Árboles



- 1) Árboles binarios: Es un árbol donde cada nodo tiene como máximo dos hijos.
- 2) Árboles AVL: Es un árbol binario equilibrado, es decir, un árbol en el cual la diferencia de altura entre los dos subárboles de cualquier nodo no excede de 1.
- 3) Árboles rojo-negro: Es un árbol binario en el que cada nodo tiene un color (rojo o negro) y se cumplen ciertas reglas para mantener el equilibrio.

Tipos de Árboles



4) Árboles B: Es un árbol en el que cada nodo puede tener varios hijos y se cumplen ciertas reglas para mantener el equilibrio.

5) Árboles B+: Es una variante de los árboles B en la cual todos los punteros apuntan a hojas, esto permite realizar búsquedas más rápidas en bases de datos.

Algunos otros tipos de árboles



- 6) Árboles de decisión: Es una estructura de datos utilizada en aprendizaje automático y minería de datos para modelar y predecir la probabilidad de una variable objetivo basándose en el valor de otras variables.
- 7) Árboles de k-d: Es un árbol binario utilizado para organizar puntos en un espacio de k dimensiones.
- 8) Árboles genealógicos: Es una representación gráfica de las relaciones de parentesco entre las personas en una familia.

Tipos de estructuras jerárquicas



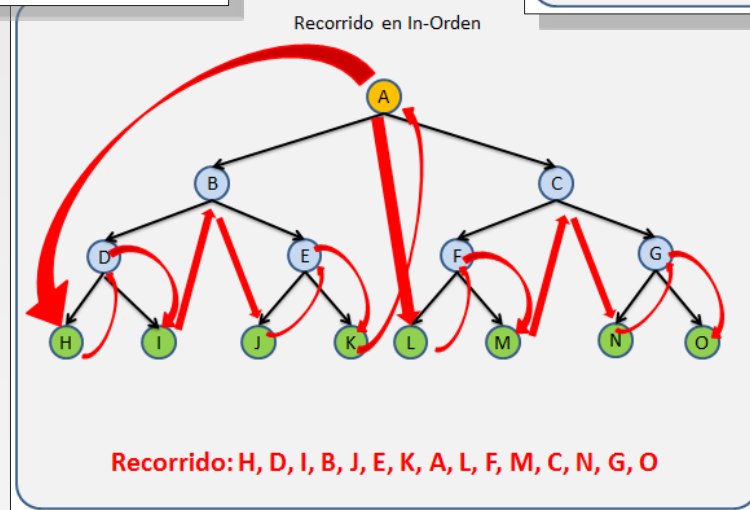
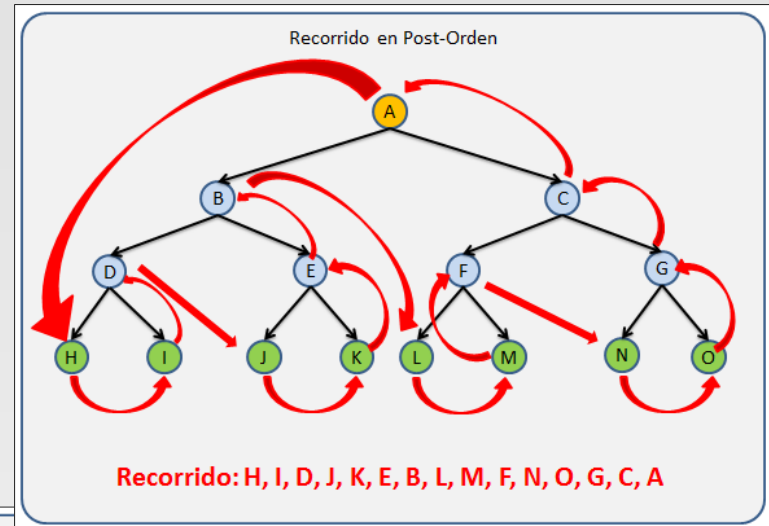
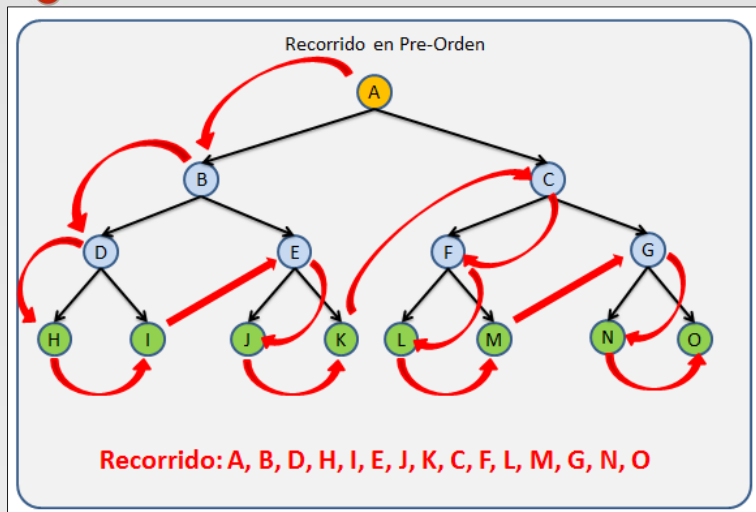
- 1) Árboles balanceados o AVL: Es un árbol binario equilibrado. Son la estructura de datos más eficiente para trabajar con la memoria principal del procesador.
- 2) Árboles semi-balanceados: Son un tipo de estructura de datos de árbol que buscan encontrar un equilibrio entre el rendimiento y el costo de mantener el equilibrio. Estos representan la estructura más eficiente para trabajar en memorias secundaria o externa.
- 3) Árboles n-arios: Son una variante de los árboles donde cada nodo puede tener un número arbitrario de hijos en lugar de solo dos o menos. Son útiles en aplicaciones donde se requiere almacenar una gran cantidad de información relacionada entre sí.

Recorrido sobre árboles



- 1) Recorrido en pre-orden: Consiste en visitar primero el nodo raíz, luego recorrer el subárbol izquierdo y finalmente el subárbol derecho.
- 2) Recorrido en in-orden: Consiste en recorrer primero el subárbol izquierdo, luego visitar el nodo raíz y finalmente el subárbol derecho.
- 3) Recorrido en post-orden: Consiste en recorrer primero el subárbol izquierdo, luego el subárbol derecho y finalmente visitar el nodo raíz.

Recorrido sobre árboles




Recorrido sobre árboles



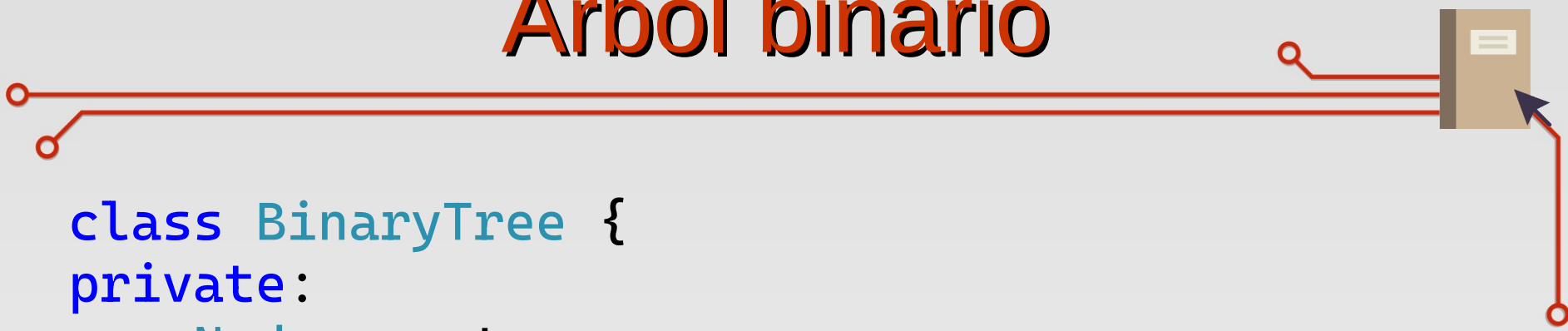
- 1) Recorrido por niveles o en anchura: Consiste en recorrer los nodos de un nivel antes de pasar al siguiente nivel. Es decir, se recorre el árbol desde la raíz hacia las hojas nivel por nivel.
- 2) Recorrido dfs (depth first search): Es una técnica de recorrido de árboles donde se recorre primero el camino más profundo y luego se retrocede.
- 3) Recorrido bfs (breadth first search): Es una técnica de recorrido de árboles donde se recorre primero el camino más amplio y luego se va profundizando.

Árbol binario



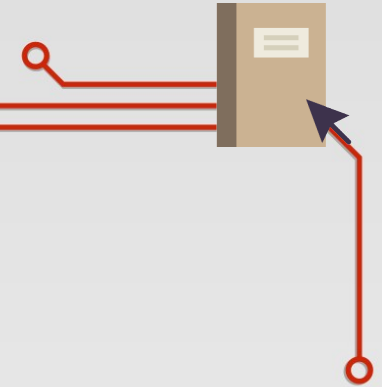
```
class Node {  
private:  
    int data;  
    Node* left, * right;  
public:  
  
    Node(int data) {  
        this->data = data;  
        left = right = NULL;  
    }  
  
    void setData(int data) { this->data = data; }  
    int getData() { return data; }  
  
    void setLeft(Node* left) { this->left = left; }  
    void setRight(Node* right) { this->right = right; }  
  
    Node* getLeft() { return left; }  
    Node* getRight() { return right; }  
};
```

Árbol binario



```
class BinaryTree {  
private:  
    Node* root;  
public:  
    BinaryTree() { root = NULL; }  
    void insert(int data);  
    Node* insertHelper(Node* node, int data);  
};
```

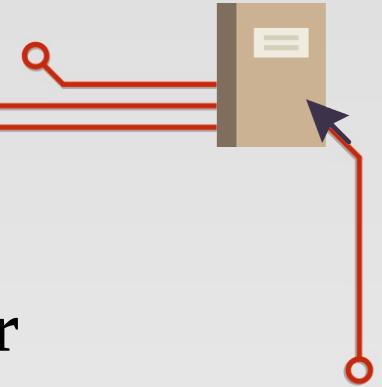
Árbol binario



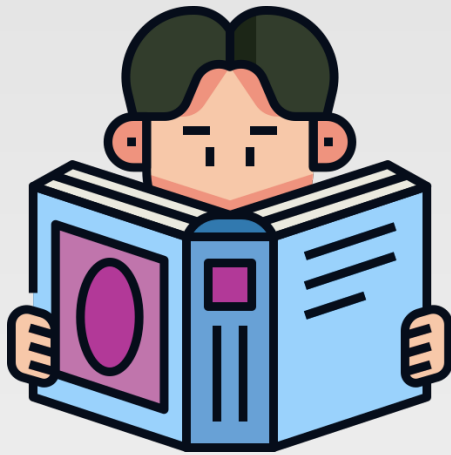
```
void BinaryTree::insert(int data) {  
    root = insertHelper(root, data);  
}
```

```
Node* BinaryTree::insertHelper(Node* node, int data) {  
    if (node == NULL) {  
        node = new Node(data);  
    }  
    else if (data <= node->getData()) {  
        node->setLeft( insertHelper(node->getLeft(), data));  
    }  
    else {  
        node->setRight( insertHelper(node->getRight(),  
data));  
    }  
    return node;  
}
```

Ejercicio de análisis



Según el código anterior

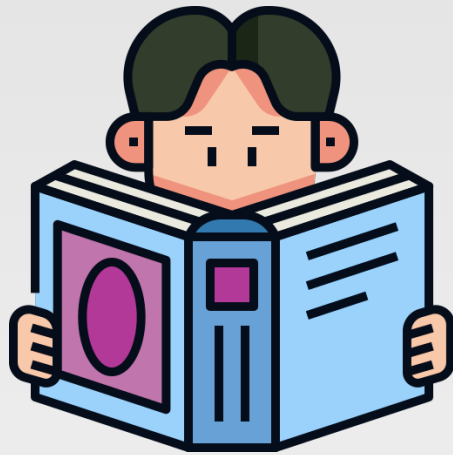


¿Cuál sería el resultado (gráficamente) del árbol si se ingresan los siguientes valores?

1, 5, 36, 3, 7, 55, 0

¿Cuál sería el resultado en pre-orden, in-orden y post-orden?

Ejercicio de análisis

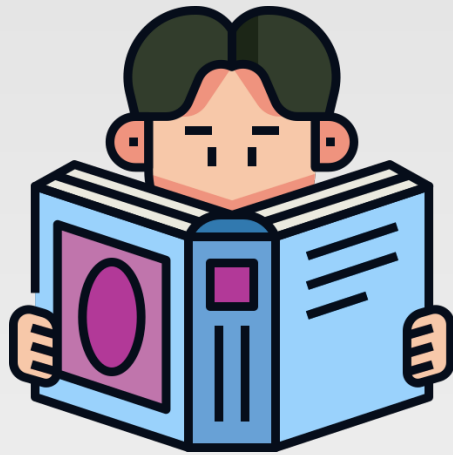


Según el código anterior

Vuelva a realizar la misma acción
(gráficamente) con los siguientes valores:

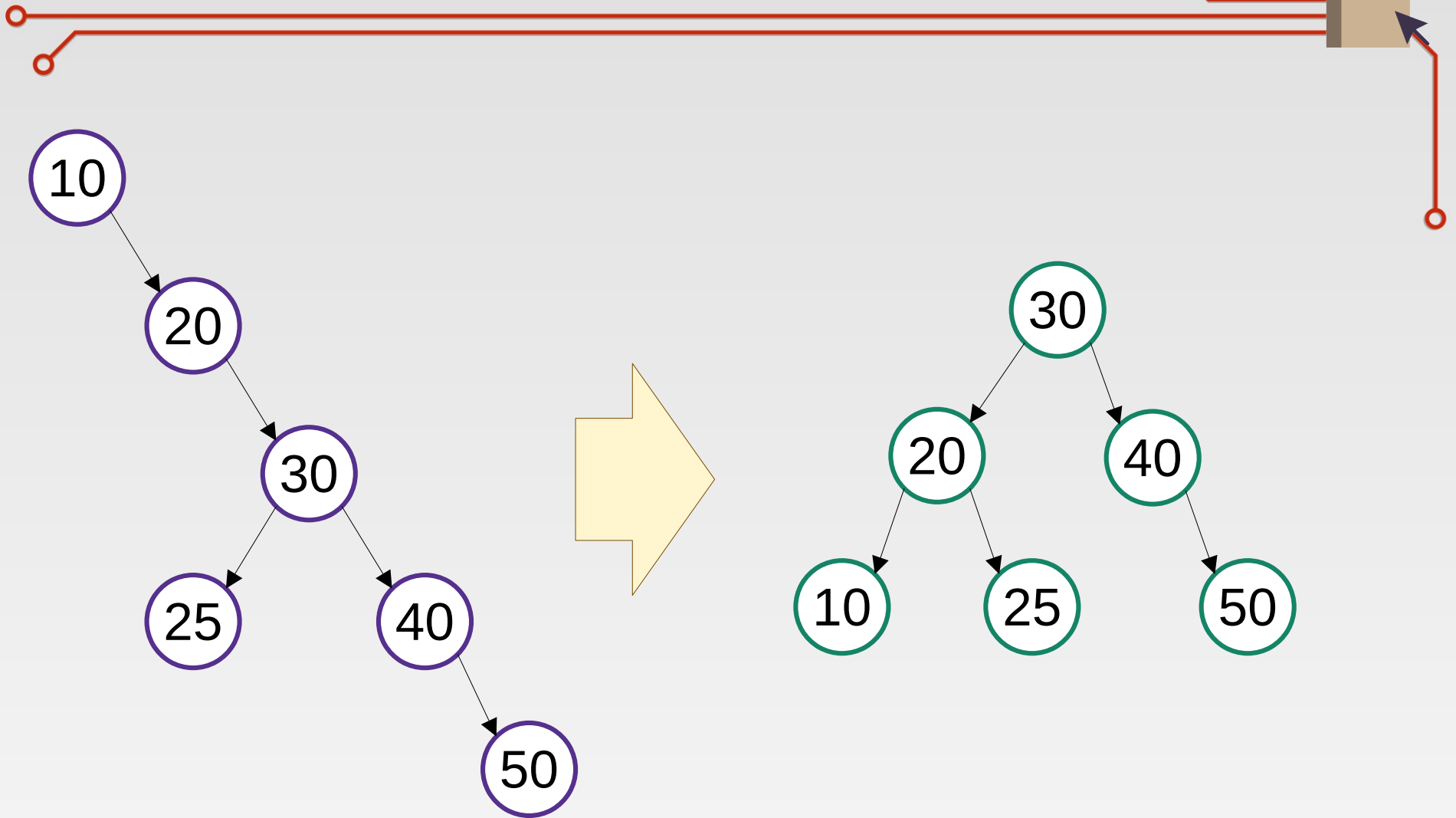
10, 20, 30, 40, 50, 25

Ejercicio de análisis



**¿Detectó algún
inconveniente en el
árbol?**

Árbol binario balanceado



Árbol binario balanceado



Algoritmo AVL

El algoritmo AVL es un algoritmo de balanceo de árboles binarios que se utiliza para garantizar que el árbol siempre esté balanceado. El algoritmo se llama así en honor a sus creadores, los matemáticos soviéticos Adelson-Velsky y Landis.

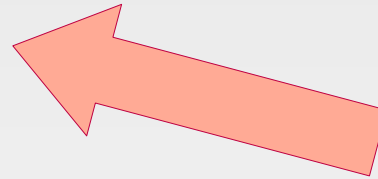
Las reglas del algoritmo AVL son las siguientes:

- 1) La diferencia en la altura entre el lado izquierdo y el lado derecho de cualquier nodo en el árbol no debe ser mayor a 1.
- 2) Cada nodo en el árbol debe tener un valor único.
- 3) Al insertar un nuevo nodo, se deben hacer rotaciones si es necesario para mantener el equilibrio del árbol.
- 4) Al eliminar un nodo, también se deben hacer rotaciones si es necesario para mantener el equilibrio del árbol.

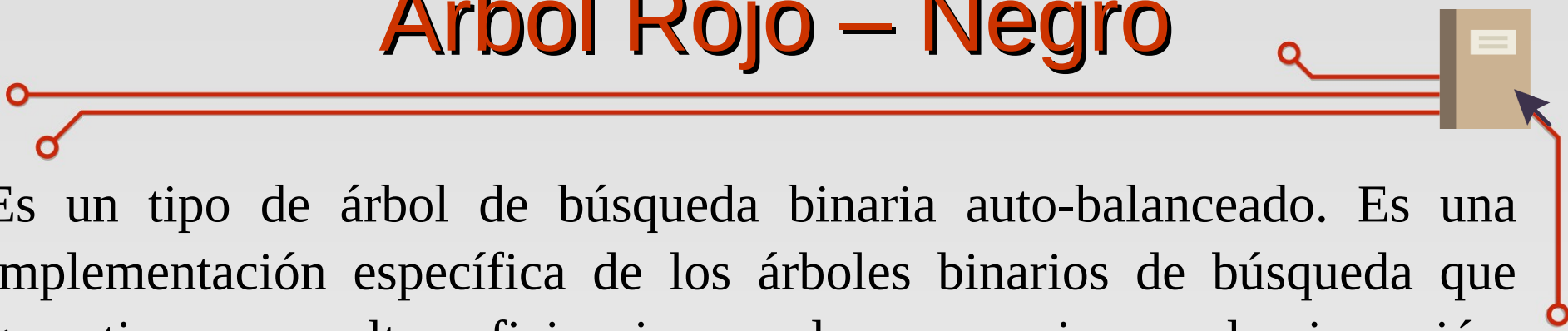
Árbol AVL



```
class NodeAVL {  
private:  
    int data;  
    NodeAVL* leftChild;  
    NodeAVL* rightChild;  
    int height;  
public:  
    NodeAVL(int data);  
    ...  
};
```



Árbol Rojo – Negro



Es un tipo de árbol de búsqueda binaria auto-balanceado. Es una implementación específica de los árboles binarios de búsqueda que garantiza una alta eficiencia en las operaciones de inserción, eliminación y búsqueda al mantener un cierto nivel de balance en el árbol.

La característica más importante de un árbol rojo-negro es que cada nodo tiene un atributo de color, que puede ser rojo o negro. A través de un conjunto de reglas sobre la asignación de colores y las rotaciones necesarias, se garantiza que el árbol siempre cumpla con las propiedades del árbol rojo-negro, lo que garantiza una alta eficiencia en las operaciones.

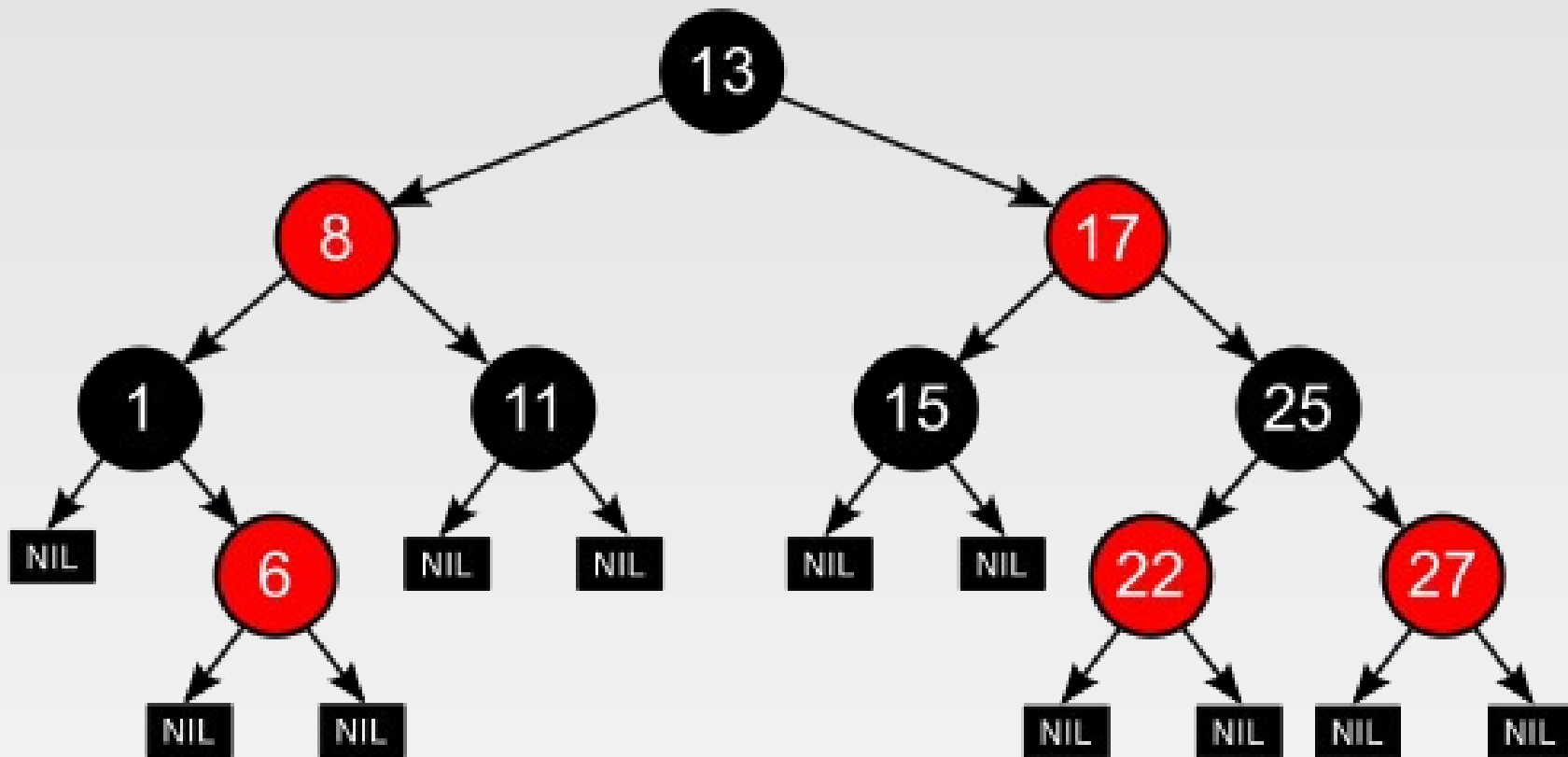
Árbol Rojo – Negro

Las reglas que se siguen para mantener un árbol rojo-negro balanceado son:

- 1) Todos los nodos son o rojos o negros
- 2) La raíz es siempre negra
- 3) Todas las hojas (nodos nulos) son negros
- 4) Si un nodo es rojo, sus hijos deben ser negros
- 5) Para cada nodo, todas las rutas desde ese nodo hasta las hojas deben tener el mismo número de nodos negros.

Al seguir estas reglas, se garantiza que el árbol no tenga más de dos niveles de diferencia entre cualquier nodo y sus hojas.

Árbol Rojo – Negro



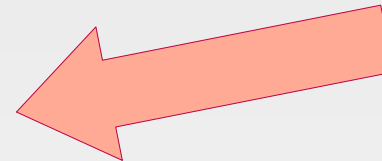
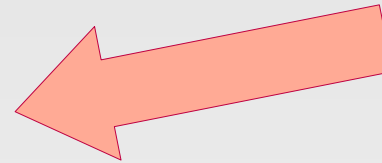
Algoritmo Árbol Rojo – Negro



- 1) Se comienza insertando el nuevo nodo como si se estuviera insertando en un BST tradicional. El nuevo nodo se colorea de rojo.
- 2) Si el nodo padre del nodo insertado es negro, no se requieren cambios adicionales. El árbol sigue cumpliendo con las reglas del árbol rojo-negro.
- 3) Si el nodo padre es rojo, entonces se deben realizar algunas operaciones de rotación y recolorado para mantener las reglas del árbol rojo-negro.
- 4) Si el nodo tío (hermano del padre) es rojo, se cambia el color del padre y del tío a negro y el abuelo a rojo. Se vuelve a evaluar el abuelo como el nuevo nodo insertado y se repiten los pasos desde el 3.
- 5) Si el nodo tío es negro, se determina si el nuevo nodo es un nodo hijo izquierdo o derecho. En función de esto, se realizarán una de dos posibles rotaciones para mantener el balance del árbol. Después de la rotación, el padre pasa a ser negro y el nodo que se convierte en el nuevo padre pasa a ser rojo.
- 6) Continuar hasta que la raíz del árbol sea negra.

Árbol Rojo – Negro

```
class NodeRN {  
private:  
    int data;  
    bool color;  
    NodeRN* left;  
    NodeRN* right;  
    NodeRN* parent;  
public:  
    Node(int data);  
    ...  
}
```

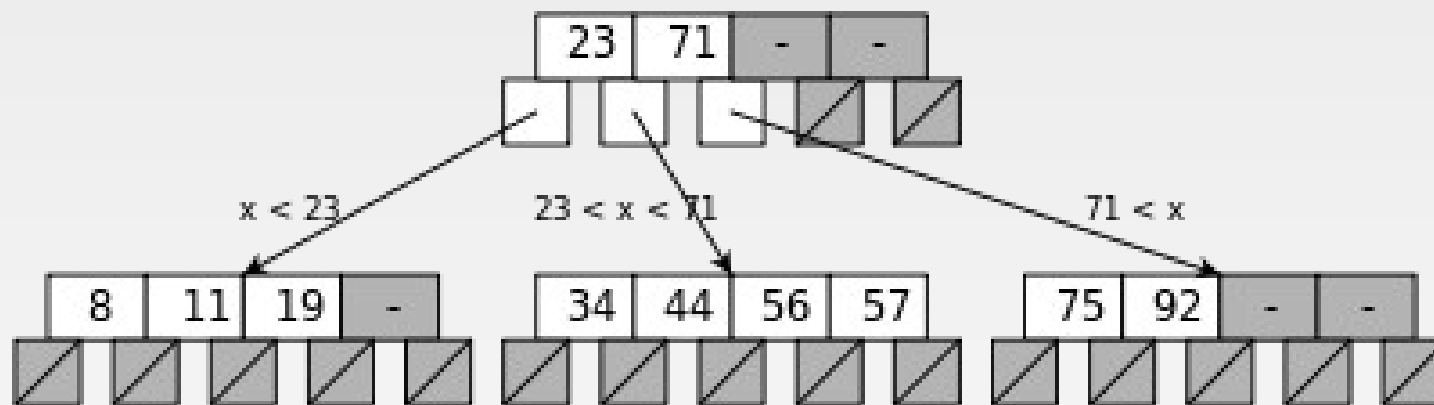


Árbol B

- Es un tipo especial de árbol binario de búsqueda (BST) que se utiliza para mantener un gran volumen de datos ordenados.
- A diferencia de un árbol binario normal, un árbol B tiene una restricción en cuanto al número de elementos que puede tener un nodo. Este número se conoce como el grado del árbol B.
- Un árbol B está diseñado para evitar la descomposición de un árbol binario normal, que puede convertirse en una lista ligada, lo cual puede ser ineficiente para buscar e insertar datos.
- Un árbol B tiene una estructura más balanceada lo que permite que las operaciones de búsqueda e inserción sean más eficientes.

Árbol B

- Los árboles B se dividen en dos tipos: árboles B y árboles B+.
- Los árboles B tienen nodos que pueden tener un número variable de claves y punteros. Cada nodo, excepto las hojas, tiene un número fijo de claves y un número más de punteros.
- Cada clave delimita un rango de valores en el sub-árbol apuntado por el puntero correspondiente.



Grado en un Árbol B



- Es un parámetro que se utiliza para controlar el número de elementos que pueden tener los nodos del árbol. El grado específico utilizado en un árbol B puede variar dependiendo de la aplicación y de la cantidad de datos que se están almacenando.
- Un grado alto permite que los nodos tengan más elementos, lo que puede mejorar la eficiencia de las operaciones de búsqueda y inserción. Sin embargo, también puede causar que los nodos se vuelven demasiado grandes y difíciles de manejar, lo que puede afectar negativamente la eficiencia del árbol.
- Por otro lado, un grado bajo puede evitar que los nodos se vuelvan demasiado grandes, pero puede causar que el árbol se descomponga y se vuelva ineficiente.
- El grado se utiliza en conjunto con otras reglas para mantener un equilibrio entre la eficiencia de las operaciones de inserción y búsqueda y evitar la descomposición del árbol. El grado es un factor importante en la estructura y el rendimiento del árbol B.

Reglas para un Árbol B



- Cada nodo, excepto las hojas, debe tener al menos grado-1 y al máximo $2 \times \text{grado}-1$ claves.
- El número de claves en un nodo debe estar en función del número de punteros. Cada nodo debe tener grado-1 claves si es un nodo interno y grado claves si es una hoja.
- Todos los nodos menos las hojas deben tener al menos grado hijos.
- Todos los nodos menos las hojas deben tener al menos grado-1 claves.
- El valor de la clave i en un nodo debe ser menor que el valor de la clave $i+1$ en el mismo nodo.
- Las hojas deben estar en el mismo nivel y deben contener todas las claves del árbol.
- Cada nodo debe apuntar al nodo con las claves más pequeñas en su sub-árbol izquierdo y al nodo con las claves más grandes en su sub-árbol derecho.
- Cada vez que un nodo se llena o queda vacío debido a una inserción o eliminación, se realizan rotaciones y fusiones para mantener la estructura del árbol cumpliendo con estas reglas.

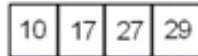
Árbol B+



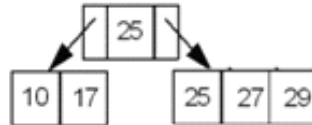
- Los árboles B+ son similares a los árboles B, pero tienen algunas diferencias importantes:
 - En lugar de almacenar las claves en cada nodo, las claves se almacenan solo en las hojas.
 - Significa que todos los nodos intermedios son iguales y solo contienen punteros a los nodos hijos.
 - Esto hace que los árboles B+ sean más adecuados para almacenar grandes cantidades de datos y para realizar búsquedas en rangos.

Árbol B+

a) INSERCIÓN: CLAVES 10, 27, 29 y 17



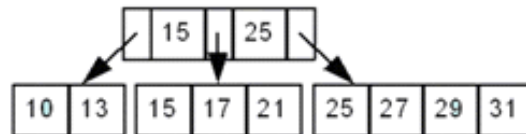
b) INSERCIÓN: CLAVE 25



c) INSERCIÓN: CLAVES 21, 15 y 31



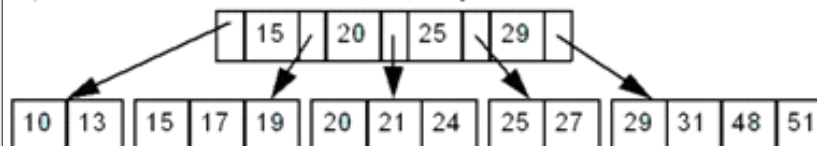
d) INSERCIÓN: CLAVE 13



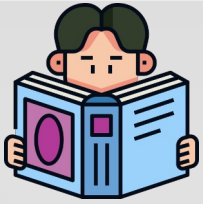
e) INSERCIÓN: CLAVE 51



f) INSERCIÓN: CLAVES 20, 24, 48 y 19



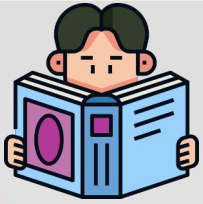
Las hojas son las que almacenan los datos



Ejercicios



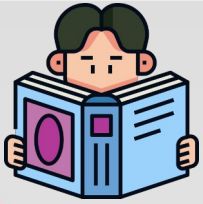
- 1) Implementar una función que dado un árbol binario de búsqueda, devuelva una lista con los valores del árbol en in-orden (izquierda, raíz, derecha).
- 2) Crear un árbol binario a partir de una lista ordenada de enteros y devolver una lista con los valores del árbol en pre-orden (raíz, izquierda, derecha).
- 3) Crear un árbol binario a partir de una lista ordenada de enteros y devolver una lista con los valores del árbol en post-orden (izquierda, derecha, raíz).
- 4) Crear un árbol binario de búsqueda e implementar una función para buscar un valor específico en el árbol.
- 5) Dado un árbol binario, implementar una función para determinar la altura del árbol. La altura de un árbol es la distancia máxima entre la raíz y una hoja.
- 6) Implementar una función para calcular la cantidad de hojas de un árbol binario.
- 7) Implementar una función para calcular la cantidad de nodos de un árbol binario.



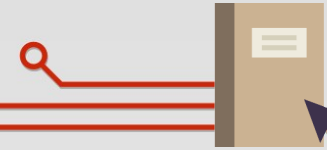
Ejercicios



- 8) Implementar una función para calcular la profundidad de un nodo específico en un árbol binario.
- 9) Dado un árbol binario, implementar una función que determine si el árbol es un árbol de búsqueda binario válido. Un árbol de búsqueda binario es válido si para cada nodo, todos los valores en su subárbol izquierdo son menores que el valor del nodo y todos los valores en su subárbol derecho son mayores que el valor del nodo.
- 10) Implementar una función para determinar si dos árboles binarios son iguales. Dos árboles son iguales si tienen la misma estructura y los mismos valores en los nodos.
- 11) Implementar una función para encontrar el ancestro común más cercano (LCA) de dos nodos en un árbol binario. El ancestro común más cercano es el nodo que tiene como descendientes a ambos nodos buscados.



Ejercicios



- 12) Escribir un algoritmo para encontrar el camino más corto entre dos nodos en un árbol binario. El algoritmo debe recibir como entrada el árbol y los identificadores de los dos nodos, y debe devolver el camino más corto entre ellos. El camino más corto se refiere al número mínimo de enlaces que se deben recorrer para llegar de un nodo a otro.
- 13) Escribir un algoritmo para encontrar el nivel más profundo en un árbol binario. El algoritmo debe recorrer el árbol en pre-orden, in-orden y post-orden, y devolver el nivel más profundo (es decir, el nivel con el mayor número de nodos) del árbol.
- 14) Escribir un algoritmo para verificar si un árbol binario es simétrico. Un árbol binario es simétrico si el sub-árbol izquierdo es un espejo del sub-árbol derecho. El algoritmo debe recorrer el árbol en pre-orden, in-orden, y post-orden, y devolver un valor booleano indicando si el árbol es simétrico o no.



Ejercicios



- 15) Escribir un algoritmo para encontrar el nodo con el valor máximo en un árbol binario de búsqueda (BST). El algoritmo debe recorrer el árbol en pre-orden, in-orden, y post-orden, y devolver el nodo que tiene el valor máximo.
- 16) Escribir un algoritmo para encontrar el predecesor in-orden de un nodo dado en un árbol binario. El predecesor in-orden es el nodo que se encuentra justo antes del nodo dado en un recorrido in-orden. El algoritmo debe recibir como entrada el árbol y el identificador del nodo, y debe devolver el predecesor in-orden.
- 17) Escribir un algoritmo para encontrar el sucesor in-orden de un nodo dado en un árbol binario. El sucesor in-orden es el nodo que se encuentra justo después del nodo dado en un recorrido in-orden. El algoritmo debe recibir como entrada el árbol y el identificador del nodo, y debe devolver el sucesor in-orden.

Bibliografía



- Cairó, O & Guardati, S. Estructuras de Datos. Tercera Edición. McGraw Hill.
- Aho, A, Hopcroft, J & Ullman, J. 1988. Estructuras de datos y algoritmos. Addison-Wesley Iberoamericana. ISBN 968-6048-19-7
- Cormen, T, et al. Introduction to Algorithms. Tercera Edición. The MIT Press Cambridge, Massachusetts.