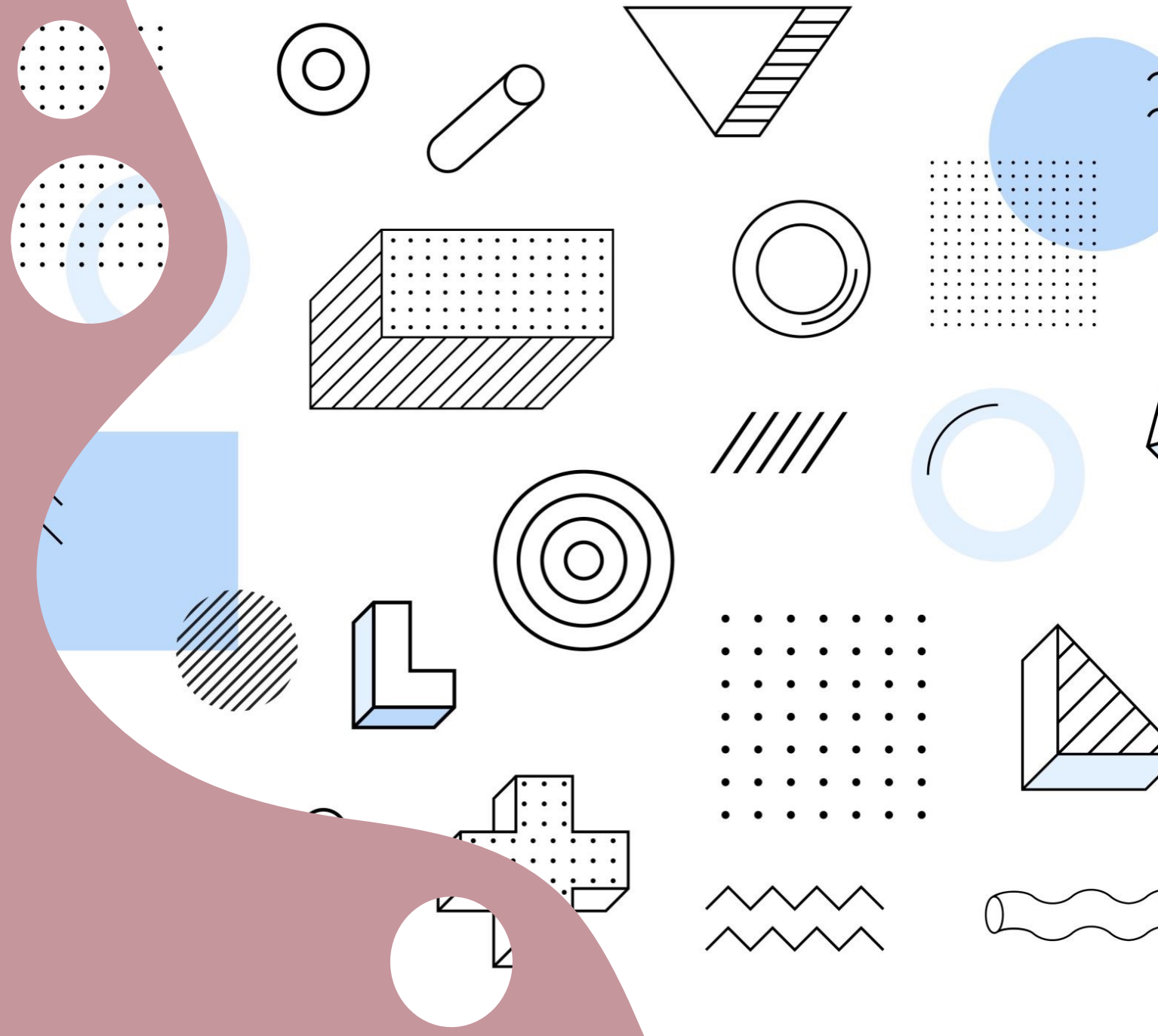
Tree - Arboles

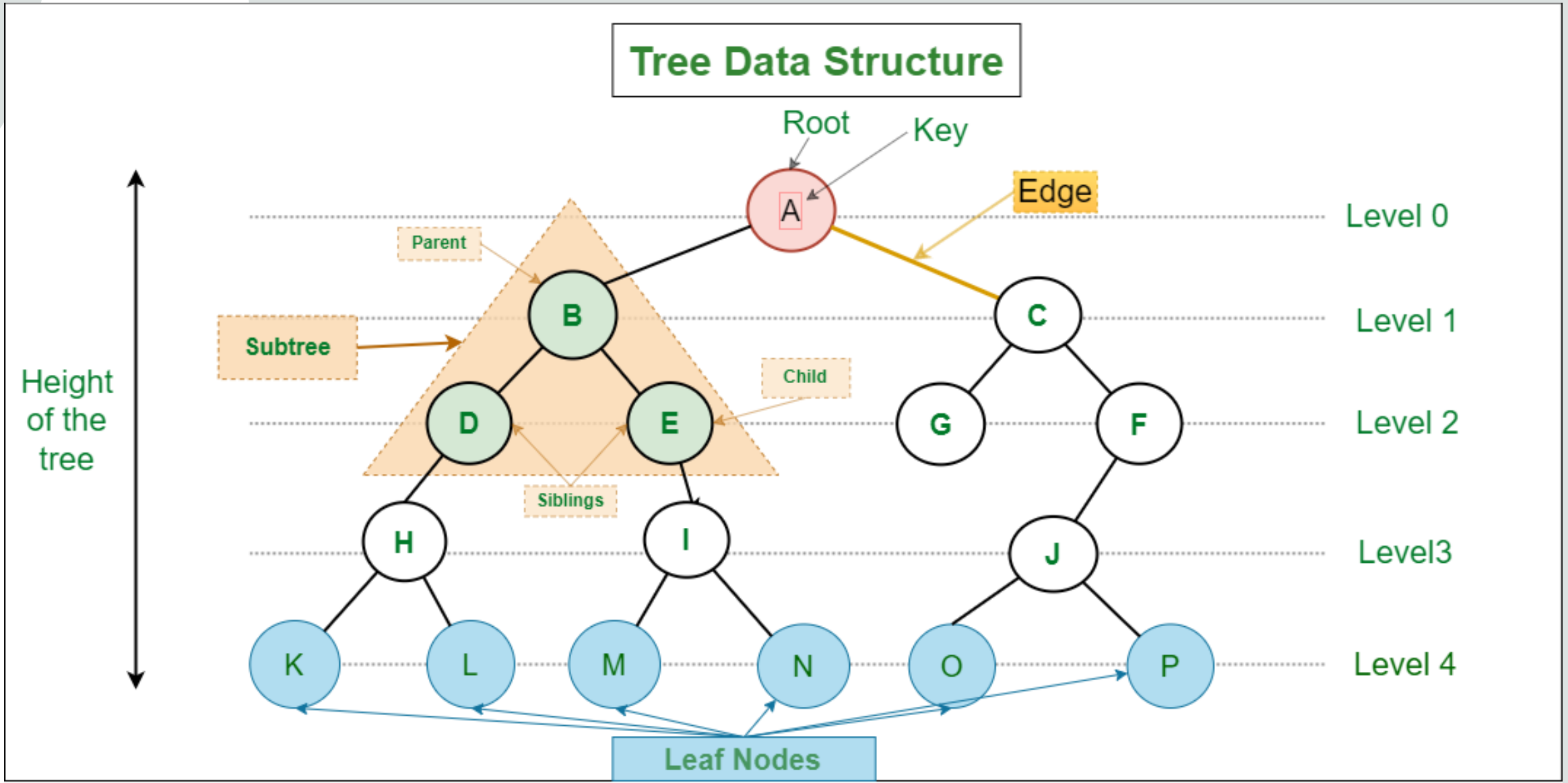
Estructura de datos



Que es?

- Estructura jerárquica no lineal, recursiva.
- Almacenar datos de forma multinivel.
- Colección de nodos conectados por aristas por medio de una relación jerarquica.
- Cada nodo puede tener múltiplos subnodos.

Representación



Propiedades

Número de aristas: Una arista se puede definir como la conexión entre dos nodos. Si un árbol tiene N nodos, tendrá (N-1) aristas. Solo hay un camino desde cada nodo a cualquier otro nodo del árbol.

Profundidad de un nodo: La profundidad de un nodo se define como la longitud del camino desde la raíz hasta ese nodo. Cada borde agrega 1 unidad de longitud al camino. Por lo tanto, también se puede definir como el número de aristas en el camino desde la raíz del árbol hasta el nodo.

Altura de un nodo: la altura de un nodo se puede definir como la longitud del camino más largo desde el nodo hasta un nodo hoja del árbol.

Altura del árbol: La altura de un árbol es la longitud del camino más largo desde la raíz del árbol hasta un nodo de hoja del árbol.

Grado de un nodo: el recuento total de subárboles adjuntos a ese nodo se denomina grado del nodo. El grado de un nodo hoja debe ser 0. El grado de un árbol es el grado máximo de un nodo entre todos los nodos del árbol.

Terminología

Nodo padre: el nodo que es un predecesor de un nodo se denomina nodo principal de ese nodo. {B} es el nodo principal de {D, E}.

Nodo hijo: el nodo que es el sucesor inmediato de un nodo se denomina nodo secundario de ese nodo. Ejemplos: {D, E} son los nodos secundarios de {B}.

Nodo Raíz: El nodo más alto de un árbol o el nodo que no tiene ningún nodo principal se llama nodo raíz. {A} es el nodo raíz del árbol. Un árbol no vacío debe contener exactamente un nodo raíz y exactamente una ruta desde la raíz a todos los demás nodos del árbol.

Nodo hoja o nodo externo: los nodos que no tienen nodos secundarios se denominan nodos hoja. {K, L, M, N, O, P} son los nodos de hoja del árbol.

Ancestro de un nodo: cualquier nodo predecesor en la ruta de la raíz a ese nodo se denomina Ancestros de ese nodo. {A,B} son los nodos antecesores del nodo {E}

Descendiente: cualquier nodo sucesor en la ruta desde el nodo hoja hasta ese nodo. {E,I} son los descendientes del nodo {B}.

Hermano: Los hijos del mismo nodo padre se llaman hermanos. {D,E} se llaman hermanos.

Nivel de un nodo: el recuento de aristas en la ruta desde el nodo raíz hasta ese nodo. El nodo raíz tiene nivel 0.

Nodo interno: Un nodo con al menos un hijo se llama Nodo interno.

Vecino de un nodo: los nodos principales o secundarios de ese nodo se denominan vecinos de ese nodo.

Subárbol: Cualquier nodo del árbol junto con su descendiente.

Representación en código

```
struct Node
{
  int data;
  struct Node *left_child;
  struct Node *right_child;
};
```

Acciones

Crear: crea un árbol en la estructura de datos.

Insertar: inserta datos en un árbol.

Buscar: busca datos específicos en un árbol para verificar si están presentes o no.

Preorder Traversal: realice el recorrido de un árbol de forma preordenada en la estructura de datos.

In Order Traversal: realice el desplazamiento de un árbol en orden.

Post Order Traversal: realiza el desplazamiento de un árbol de manera posterior a la orden.

Aplicaciones

- Binary Search Tree
- Heap
- B-Tree
- Syntax Tree
- K-D Tree
- Trie
- Suffix Tree
- Spanning Trees

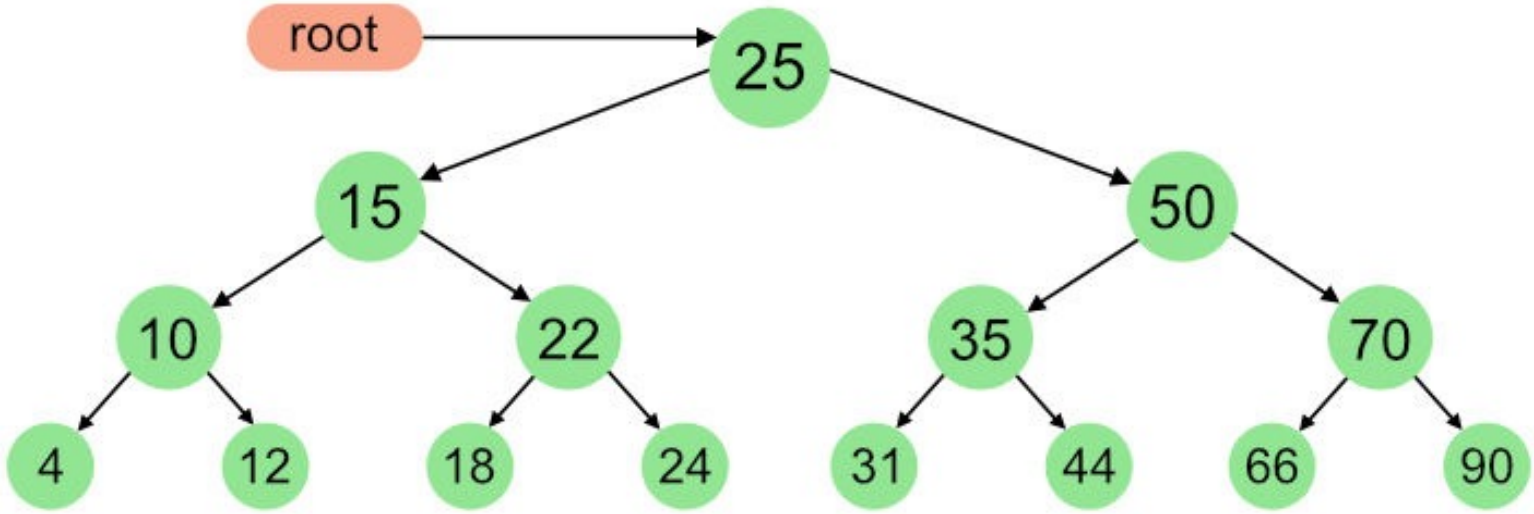
Recorridos

- InOrder
- PreOrder
- PostOrder

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



InOrder

```
/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct Node* node)
{
     if (node == NULL)
          return;

     /* first recur on left child */
     printInorder(node->left);

     /* then print the data of node */
     cout << node->data << " ";

     /* now recur on right child */
     printInorder(node->right);
}
```

PreOrder

```
/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct Node* node)
{
    if (node == NULL)
        return;

    /* first print data of node */
    cout << node->data << " ";

    /* then recur on left subtree */
    printPreorder(node->left);

    /* now recur on right subtree */
    printPreorder(node->right);
}
```

PostOrder

```
/* Given a binary tree, print its nodes according to the
"bottom-up" postorder traversal. */
void printPostorder(struct Node* node)
{
    if (node == NULL)
        return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

    // now deal with the node
    cout << node->data << " ";
}
```