Arboles

Por Ariel Parra

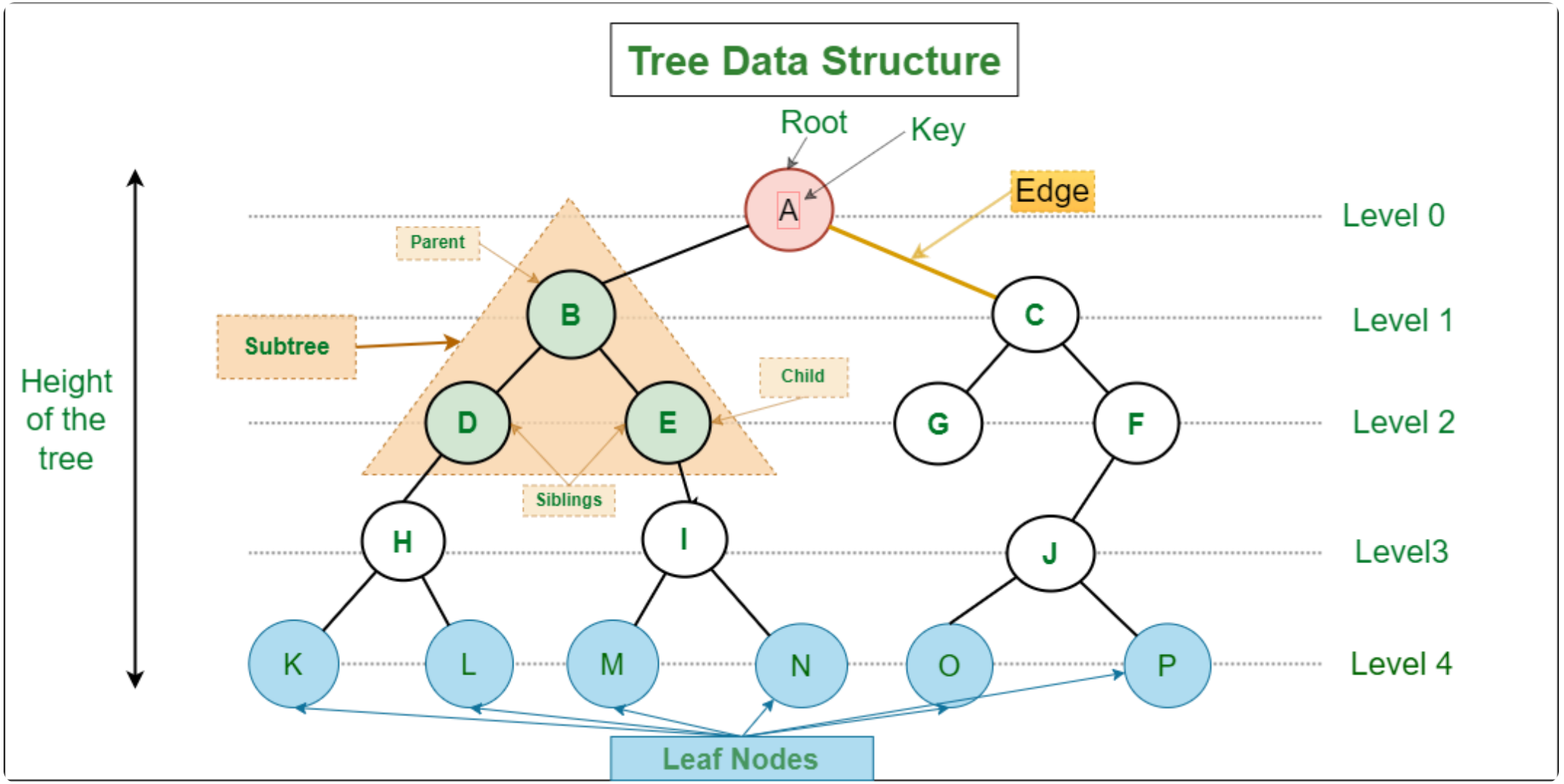
[Γα=Ω5]

¿Qué es un Arbol o un tree en c++?

Es una estructura jerárquica de datos **no lineales** que se utiliza para representar y organizar datos de una manera que es fácil de navegar y buscar. Es una colección de nodos que están conectados por bordes y tiene una relación jerárquica (padre e hijos) entre nodos.

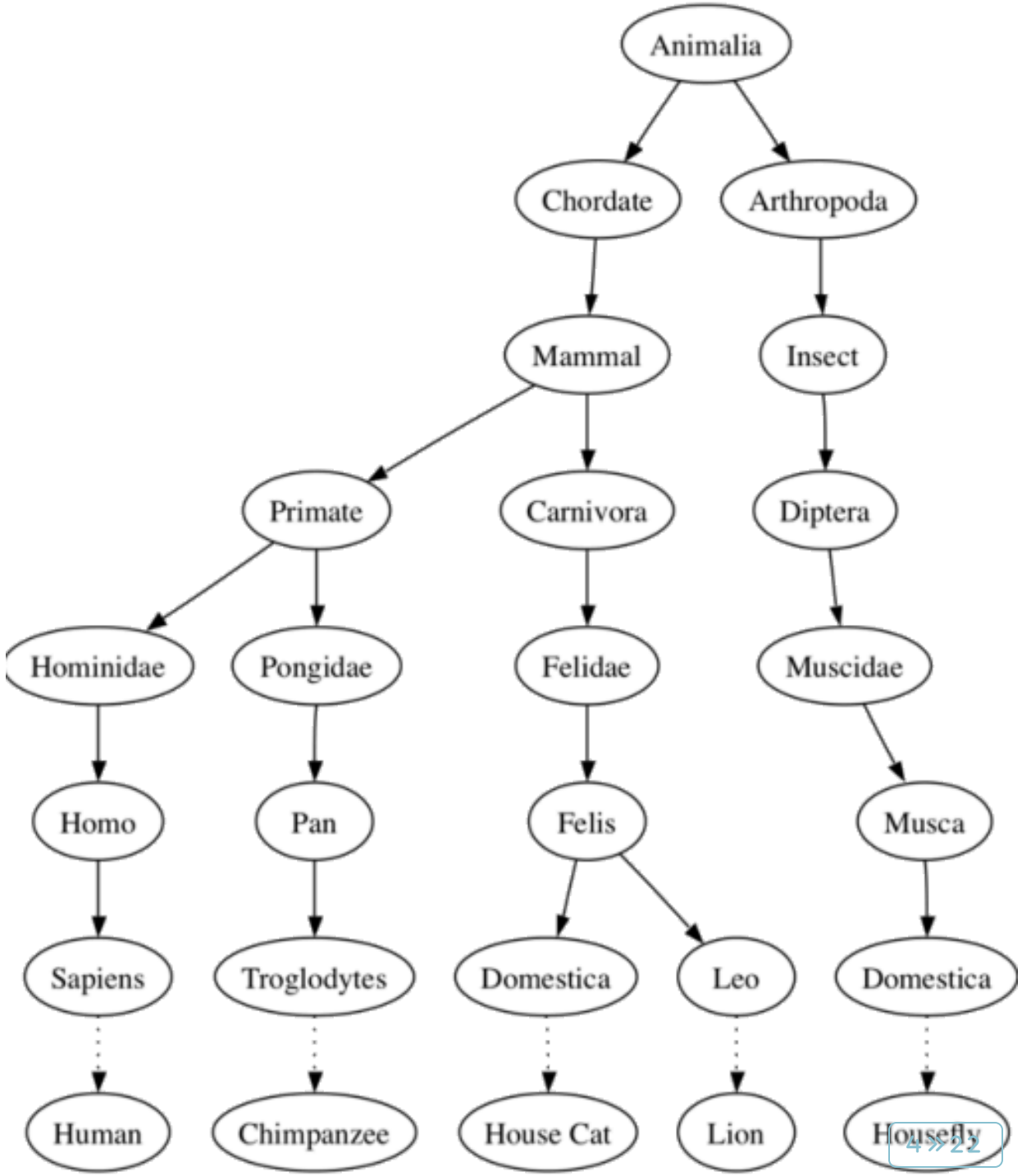
El nodo más alto del árbol se llama la **raíz**, y los nodos debajo se llaman **nodos hijos**. Cada nodo puede tener múltiples hijos, y estos nodos hijos también pueden tener sus propios hijos, formando una estructura recursiva.

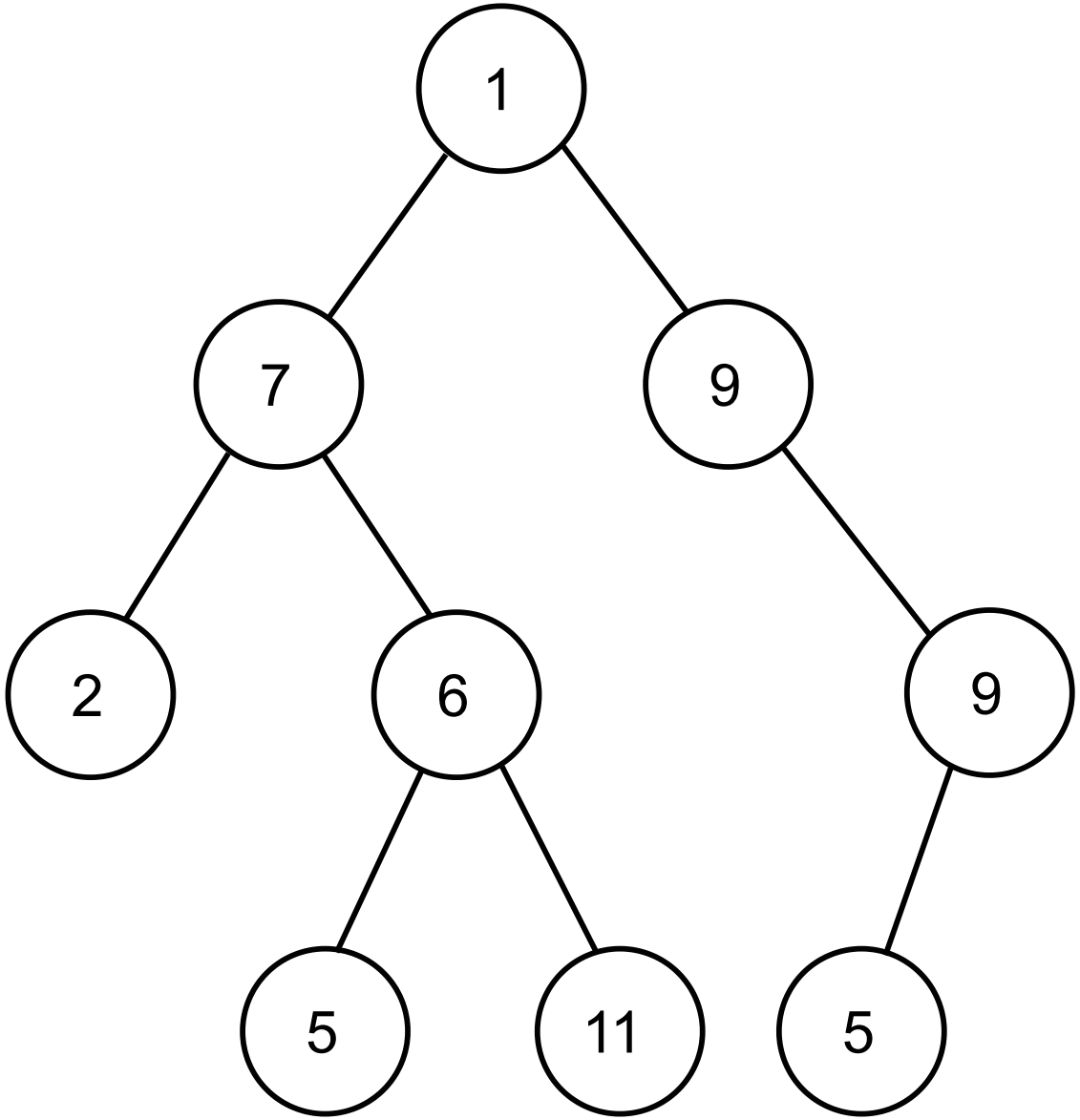
terminologia



Ejercicio en clase

- ¿Cuántos nodos padres tiene el árbol?
- ¿Cuántos hijos tiene el nodo "Mammal"?
- ¿Cuál es la rama más grande del árbol?
- ¿Qué profundidad tiene el nodo "Pan"?
- ¿Cuál tiene mayor altura entre los nodos "Mammal" e "Insecto"?
- ¿Cuántos nodos tiene el subárbol "Felidae"?





Binary tree

El árbol binario se define como una estructura de datos de árboles donde cada nodo tiene maximo 2 hijos. Denominamos el a los hijos como izquierdo y derecho (left and right).

```
struct node {
        int data;
        node* left;
        node* right;
};
```

Función recursiva para determinar la altura de un binary tree

```
int height(Node* node){
        if (node == nullptr) return 0;

        // height = 1 + max of left height
        // and right heights
        return 1 + max(height(node->left), height(node->right));
}// O(n) (cada nodo es visitado una sola vez)
```

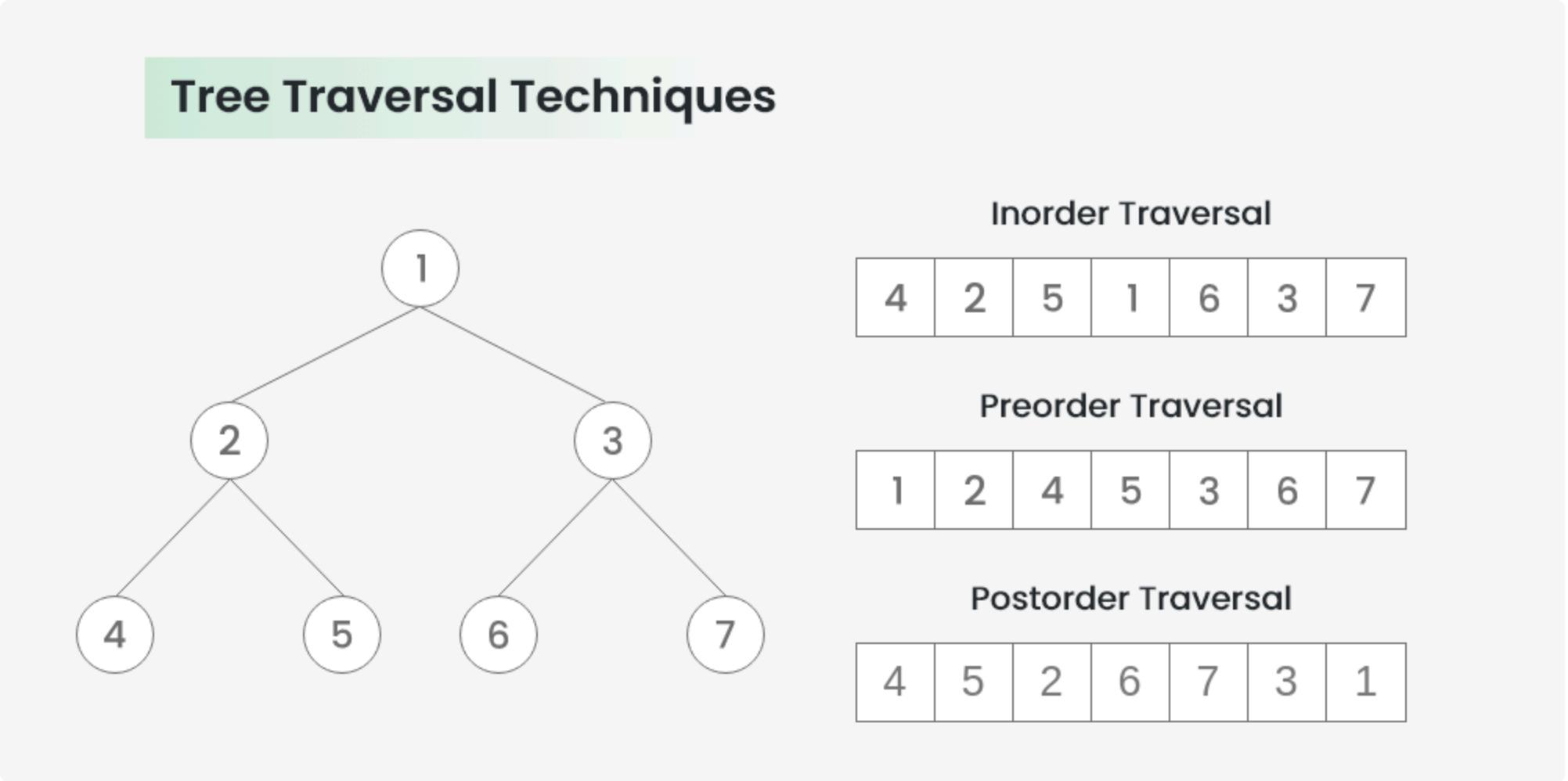
Determinar el balance de un binary tree

```
int isBalanced(Node* root){
    if (root == nullptr) return 0;
    //left subtree
    int lh = isBalanced(root->left);
    if (lh == -1) return -1;
    //right subtree
    int rh = isBalanced(root->right);
    if (rh == -1) return -1;
    if (abs(lh - rh) > 1)
        return -1;
    else
        return max(lh, rh) + 1;
}// O(n) (cada nodo es visitado una sola vez)
```

Ejercicio en clase

```
int main(){
        Node* root = new Node(1);
        root->left = new Node(2);
        root->right = new Node(3);
        root->left->left = new Node(4);
        root->left->right = new Node(5);
        root->left->left->left = new Node(8);
        if (isBalanced(root))
                cout << "Tree is balanced";
        else
                cout << "Tree is not balanced";
        return 0;
}
```

Recorrido de arbol (Traversal)



Inorder

```
void printInorder(struct Node* node){
    if (node == nullptr) return;
    printInorder(node->left);
    cout << node->data << " ";
    printInorder(node->right);
}
```

Preorder

```
void printPreorder(struct Node* node){
    if (node == nullptr) return;
    cout << node->data << " ";
    printPreorder(node->left);
    printPreorder(node->right);
}
```

Postorder

```
void printPostorder(struct Node* node){
    if (node == nullptr) return;
    printPostorder(node->left);
    printPostorder(node->right);
    cout << node->data << " ";
}
```

Infix expression	postfix expression	prefix expression
A + B	AB +	+ AB
A + B * C	A B C * +	+ A * B C

El recorrido prefix, infix y postfix se entiende mejor al visualizar operaciones con números. No usar el formato infix se prefiere computacionalmente, ya que permite ahorrar paréntesis.

> Las operaciones en formato postfix también se conocen como Notación Polaca Inversa (RPN)

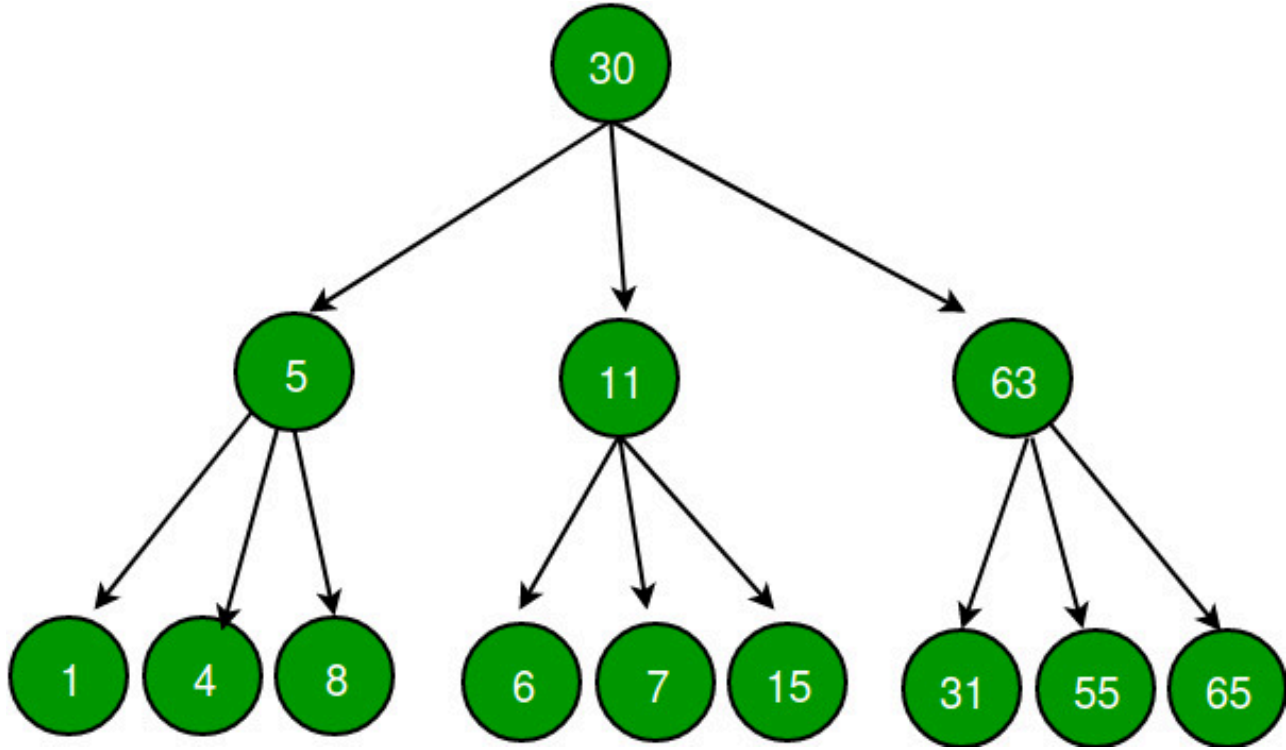
Level Order Traversal

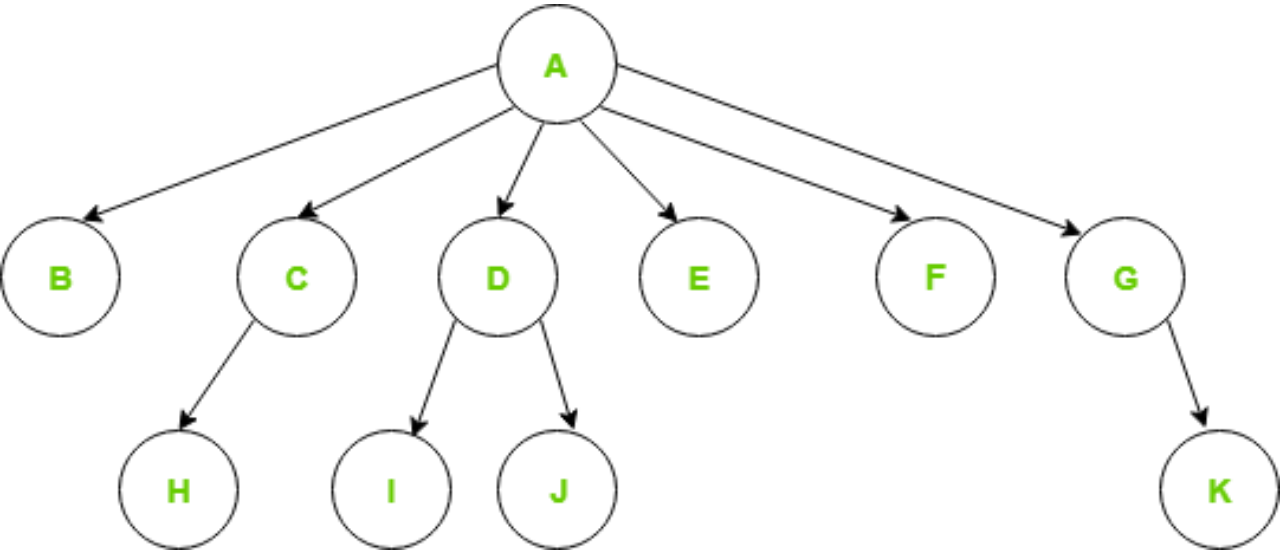
```
void printLevelOrder(node* root){
    int h = height(root);
    int i;
    for (i = 1; i <= h; i++)
        printCurrentLevel(root, i);
}
void printCurrentLevel(node* root, int level){
    if (root == nullptr)
        return;
    if (level == 1)
        cout << root->data << " ";
    else if (level > 1) {
        printCurrentLevel(root->left, level - 1);
        printCurrentLevel(root->right, level - 1);
    }
}
```

Ternary tree

El árbol ternarios es una estructura de datos de árboles en la que cada nodo tiene a la mayoría de tres nodos infantiles, generalmente distinguidos como “izquierda”, “medio” y “derecho”.

```
struct node {
    int data;
    node* left;
    node* center;
    node* right;
};
```





N-ary Tree

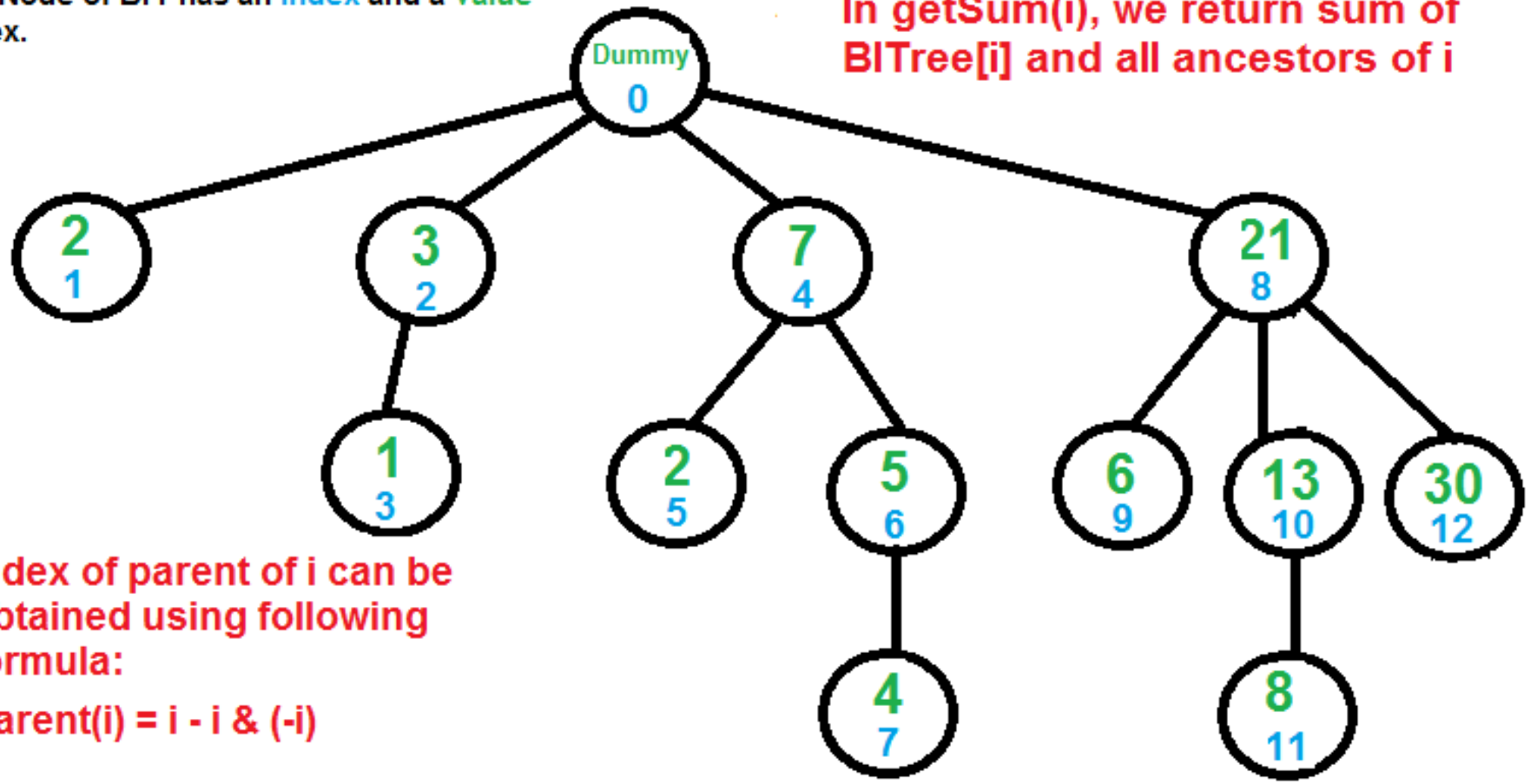
Los árboles N-arios o árboles genéricos son una colección de nodos donde cada nodo es una estructura de datos que consiste en registros y una lista de referencias a sus hijos (no se permiten referencias duplicadas).

```
struct Node{
        int data;
        Node *firstChild;
        Node *nextSibling;
};
```

Binary Indexed Tree (BIT) / Fenwick Tree

Un árbol de Fenwick o Árbol Binario Indexado (BIT) es una estructura de datos que permite actualizar valores y calcular sumas prefijas de manera eficiente en un arreglo de valores. El Árbol Binario Indexado se representa como un arreglo, denominado BITree[]. Cada nodo del Árbol Binario Indexado almacena la suma de algunos elementos del arreglo de entrada. El tamaño del Árbol Binario Indexado es igual al tamaño del arreglo de entrada, denotado como n. En el código a continuación, usamos un tamaño de n+1 para facilitar la implementación.

Every Node of BIT has an Index and a Value at index.



In getSum(i), we return sum of BITree[i] and all ancestors of i

Index of parent of i can be obtained using following formula:

parent(i) = i - i & (-i)

The above formula basically removes the last set bit from i. For example, if i = 12, then parent(i) is 8

			0	1	2	3	4	5	6	7	8	9	10	11
Input Array:	arr[0..n-1]	=	{2,	1,	1,	3,	2,	3,	4,	5,	6,	7,	8,	9}
BI Tree Array:	BITree[1..n]	=	{2,	3,	1,	7,	2,	5,	4,	21,	6,	13,	8,	30}
			1	2	3	4	5	6	7	8	9	10	11	12

View of Binary Indexed Tree to unserstand getSum() operation

```cpp
// Returns sum of arr[0..index].
int getSum(const vector<int>& BITree, int index) {
    int sum = 0; // Initialize result

    // index in BITree[] is 1 more than the index in arr[]
    index = index + 1;

    // Traverse ancestors of BITree[index]
    while (index > 0)
    {
        // Add current element of BITree to sum
        sum += BITree[index];

        // Move index to parent node in getSum View
        index -= index & (-index);
    }
    return sum;
}
```

```cpp
// Updates a node in Binary Index Tree (BITree) at given index
// in BITree. The given value 'val' is added to BITree[i] and
// all of its ancestors in tree.
void updateBIT(vector<int>& BITree, int index, int val) {
    // index in BITree[] is 1 more than the index in arr[]
    index = index + 1;

    // Traverse all ancestors and add 'val'
    while (index < BITree.size())
    {
        // Add 'val' to current node of BI Tree
        BITree[index] += val;

        // Update index to that of parent in update View
        index += index & (-index);
    }
}
```

```cpp
// Constructs and returns a Binary Indexed Tree for given array
vector<int> constructBITree(const vector<int>& arr) {
    int n = arr.size();
    vector<int> BITree(n + 1, 0);
    // Store the actual values in BITree[] using update()
    for (int i = 0; i < n; i++)
        updateBIT(BITree, i, arr[i]);
    return BITree;
}
```

```cpp
int main() {
    vector<int> freq = {2, 1, 1, 3, 2, 3, 4, 5, 6, 7, 8, 9};
    vector<int> BITree = constructBITree(freq);
    cout << "Sum of elements in arr[0..5] is "<< getSum(BITree, 5);
    freq[3] += 6;
    updateBIT(BITree, 3, 6); // Update BIT for above change in arr[]
    cout << "\nSum of elements in arr[0..5] after update is " << getSum(BITree, 5);
    return 0;
}
```

Referencias

- GeeksforGeeks. (2024). *Introduction to Tree Data Structure*. Recuperado de https://www.geeksforgeeks.org/introduction-to-tree-data-structure-and-algorithm-tutorials/
- GeeksforGeeks. (2024). *Tree Data Structure*. Recuperado de https://www.geeksforgeeks.org/tree-data-structure/
- GeeksforGeeks. (2024). *Tree Traversal Techniques*. Recuperado de https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/
- GeeksforGeeks. (2024). *Introduction to Binary Tree*. Recuperado de https://www.geeksforgeeks.org/binary-tree-data-structure/
- GeeksforGeeks. (2023). *Binary Indexed Tree or Fenwick Tree*. Recuperado de https://www.geeksforgeeks.org/binary-indexed-tree-or-fenwick-tree-2/