THE BINARY TREE

THE TREE

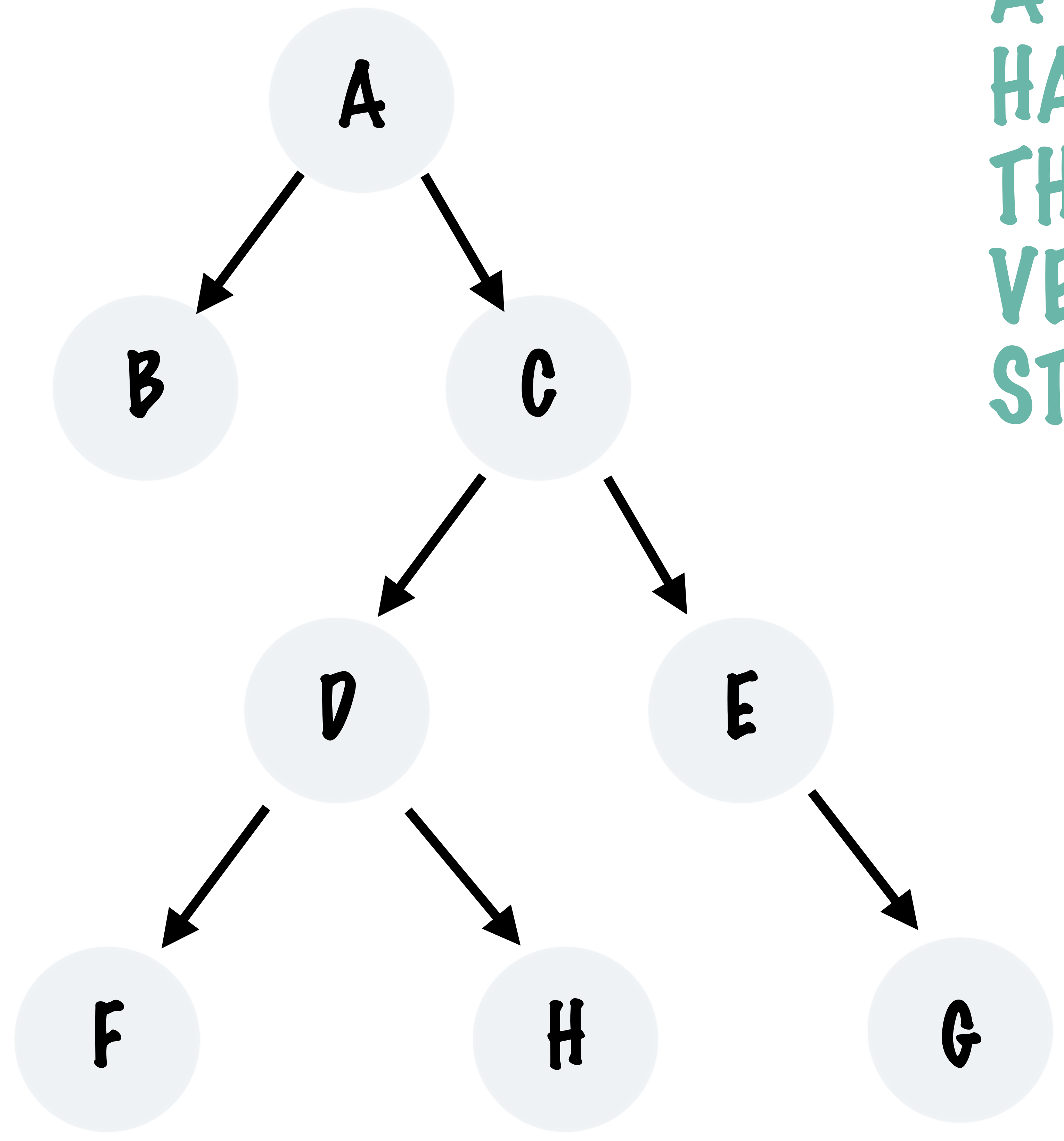
A TREE IS A DATA STRUCTURE WHICH IS MADE UP OF NODES. EACH NODE CAN POINT TO A NUMBER OF NODES, NOT JUST ONE

UNLIKE STACKS, QUEUES, LINKED LISTS ETC. THE ORDER OF THE ELEMENTS IS NOT IMPORTANT IN A TREE.

IT'S A NON-LINEAR DATA STRUCTURE

A TREE IS TYPICALLY USED TO REPRESENT HIERARCHICAL INFORMATION

THE BINARY TREE

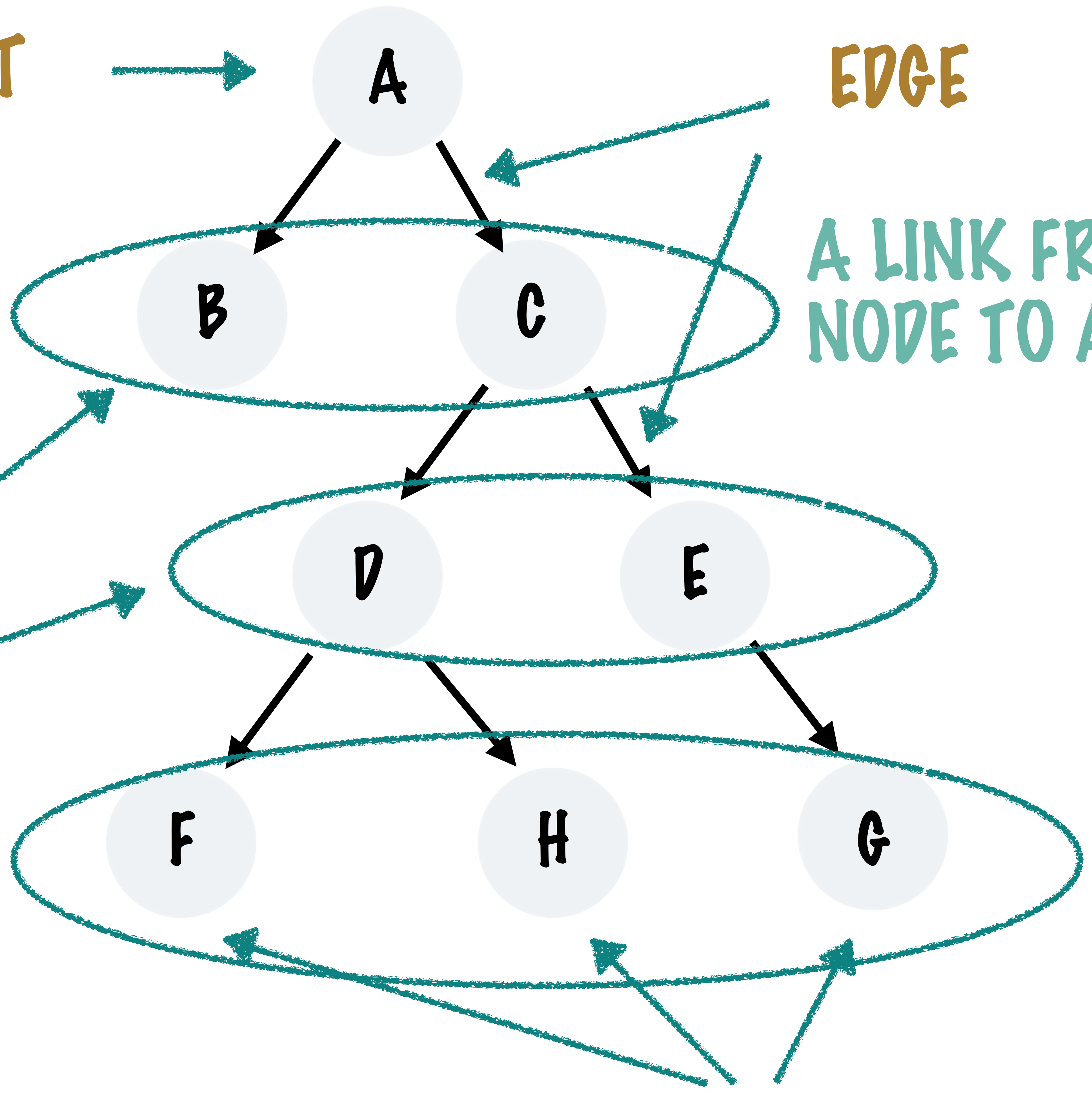


A GENERAL TREE DATA STRUCTURE CAN HAVE ANY NUMBER OF CHILDREN BUT THESE TREES ARE LESS USEFUL AND NOT VERY COMMONLY USED AS A DATA STRUCTURE

IN A BINARY TREE EACH NODE CAN HAVE 0, 1, OR 2 CHILDREN

WE'LL BE FOCUSING EXCLUSIVELY ON BINARY TREES AND IT'S VARIATIONS

THE BINARY TREE



ROOT

A NODE WITH NO PARENTS, EVERY TREE HAS EXACTLY ONE ROOT

EDGE

A LINK FROM A PARENT NODE TO A CHILD NODE

SIBLINGS

NODES AT THE SAME LEVEL IN THE TREE

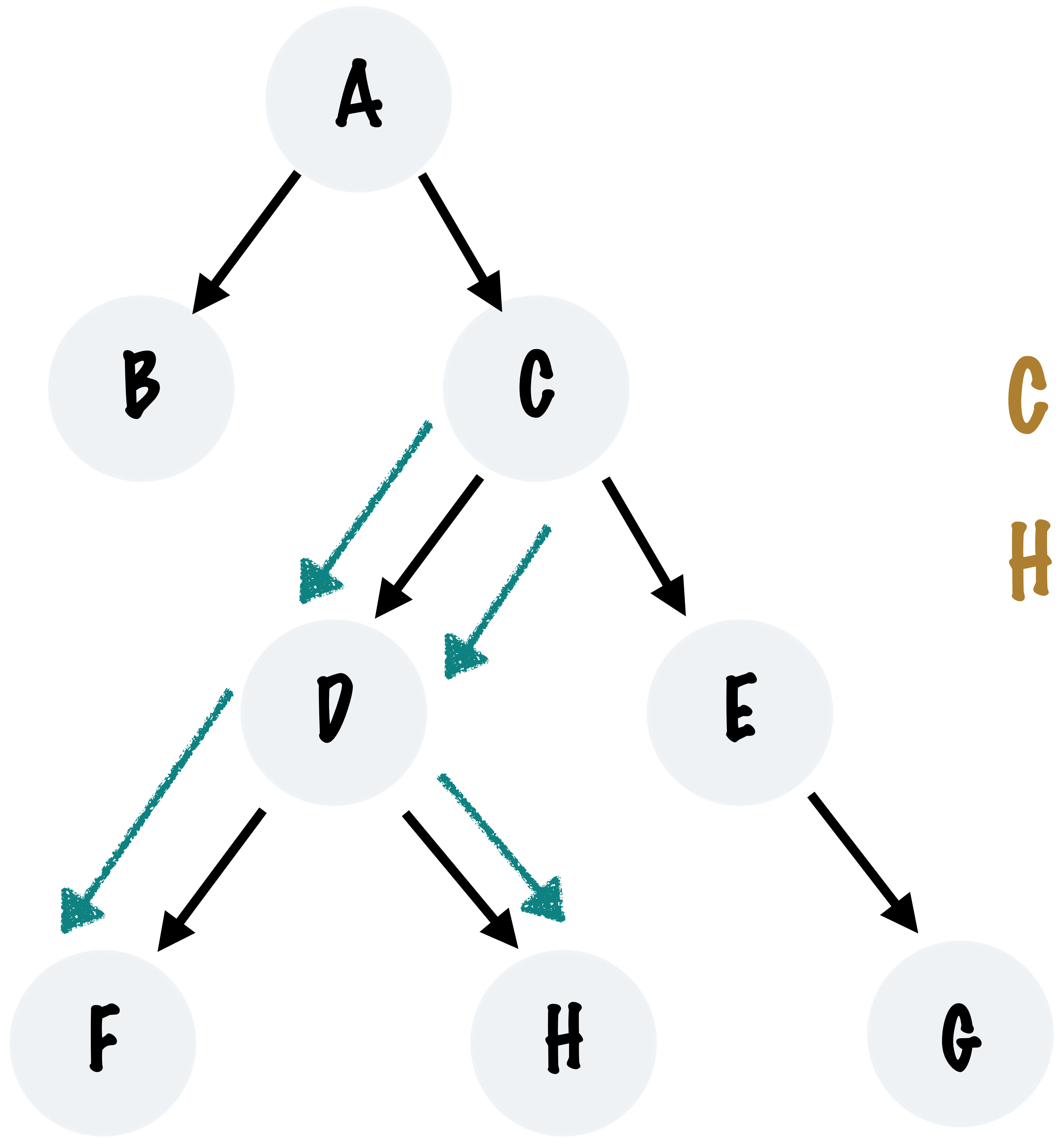
LEAF

A NODE WITH NO CHILDREN

THE BINARY TREE

THE ROOT NODE A IS AN ANCESTOR OF EVERY NODE

EVERY NODE IS A DESCENDENT OF THE ROOT NODE



C IS AN ANCESTOR OF H

H IS A DESCENDENT OF C

C IS AN ANCESTOR OF F

F IS A DESCENDENT OF C

A TREE NODE

```
public static class Node<T> {
    private T data;
    private Node<T> leftChild;
    private Node<T> rightChild;

    public Node(T data) {
        this.data = data;
    }

    public T getData() {
        return data;
    }

    public Node<T> getLeftChild() {
        return leftChild;
    }

    public void setLeftChild(Node<T> leftChild) {
        this.leftChild = leftChild;
    }

    public Node<T> getRightChild() {
        return rightChild;
    }

    public void setRightChild(Node<T> rightChild) {
        this.rightChild = rightChild;
    }
}
```

A GENERIC TREE NODE, CAN HOLD DATA OF OF ANY TYPE

A NODE CAN HAVE A MAXIMUM OF 2 CHILDREN

A WHOLE BUNCH OF HELPER METHODS TO WORK WITH THE NODE