UNIT 5 · NON-LINEAR STRUCTURES

# 16. Trees and General Trees

CSC211 · Algorithms and Data Structures

COMSATS University Islamabad, Lahore Campus

# Agenda

N

### Why Non-Linear Structures?

One element can branch into many children at once

T

### Tree Concept & Terminology

Root, parent, child, sibling, leaf, depth, height

B

### Binary Trees

At most two children — full, complete, and skewed shapes

[]

### Array vs. Linked Representation

Two ways to store a binary tree in memory

G

### General Tree -> Binary Tree

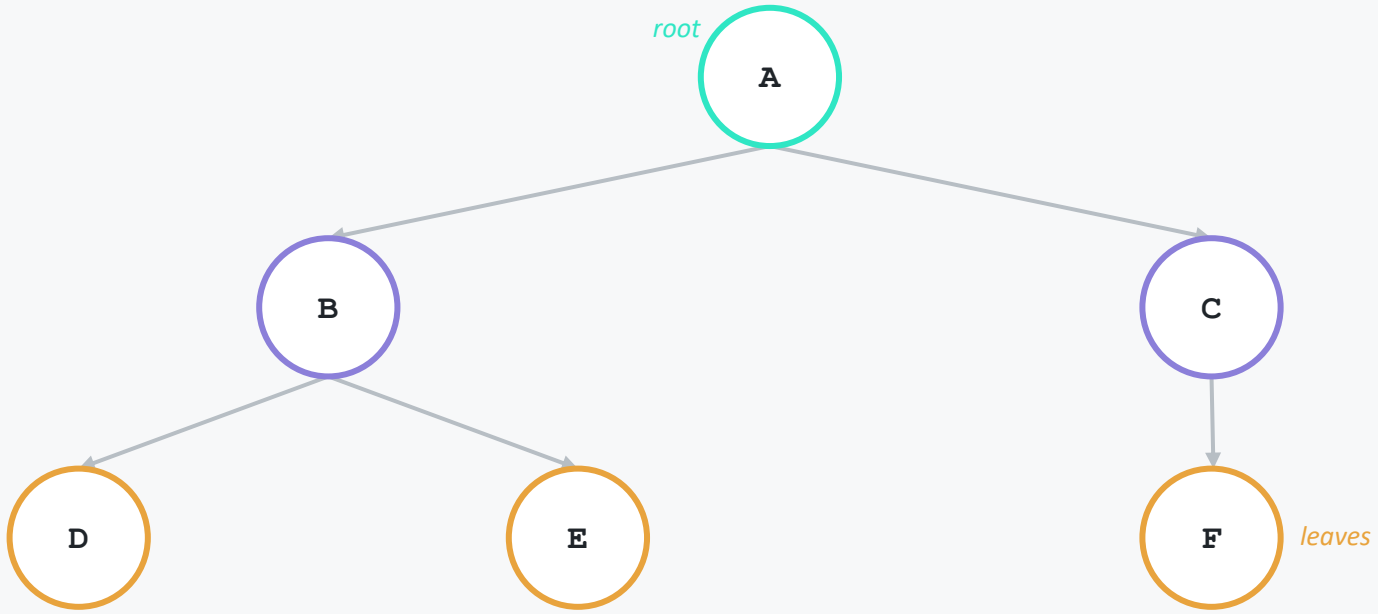
The first-child/next-sibling (FCNS) trick

W

### Real-World Applications

File systems, org charts, the DOM, compilers

# The Tree Concept and Terminology




**Root**
No parent — the top (A)

**Leaf**
No children (D, E, F)

**Sibling**
Same parent (B, C)

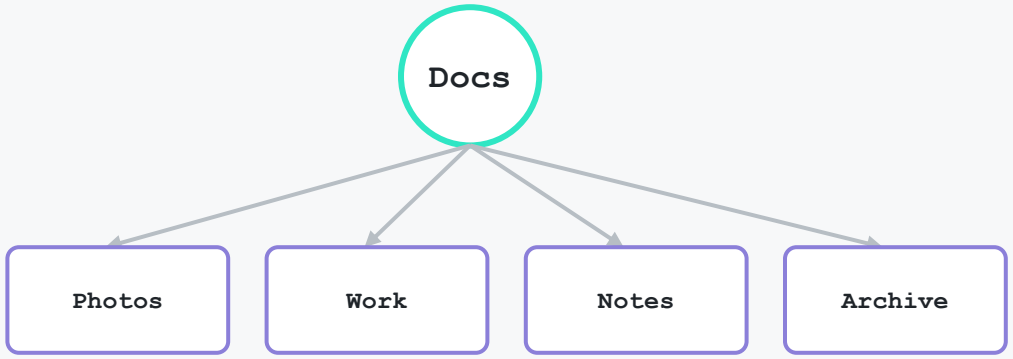
**Depth**
Edges from root down

**Height**
Longest root-to-leaf path (2)

*A general tree places no limit on how many children a node can have — binary trees add one restriction to simplify analysis.*

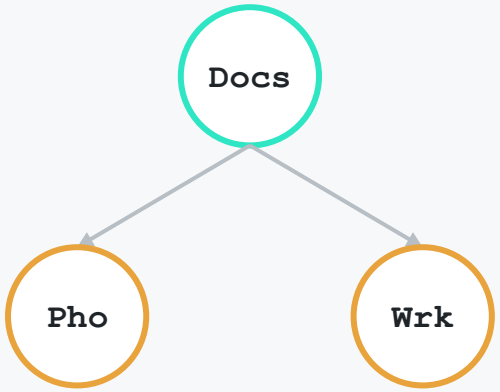
SIDE BY SIDE

# General Tree vs. Binary Tree




**General tree — any number of children**

Documents has FOUR children — perfectly normal for a folder.




**Binary tree — at most 2 children**

No third or fourth pointer to use — needs a different strategy entirely.

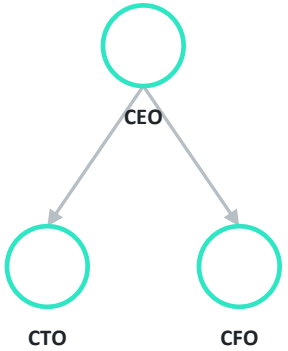
This isn't cosmetic: it's the reason binary trees need a completely different strategy (first-child/next-sibling, later this lecture) to represent a general tree's unlimited branching at all.

ONE POINTER ASSIGNMENT AT A TIME

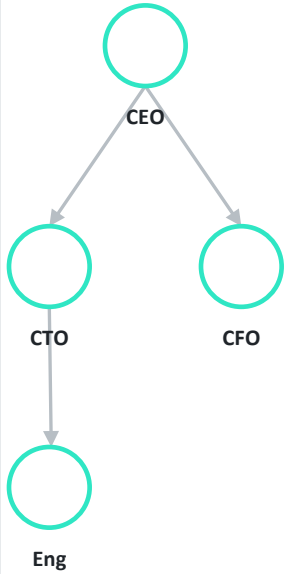
# Building a Tree, Node by Node

**Step 1**

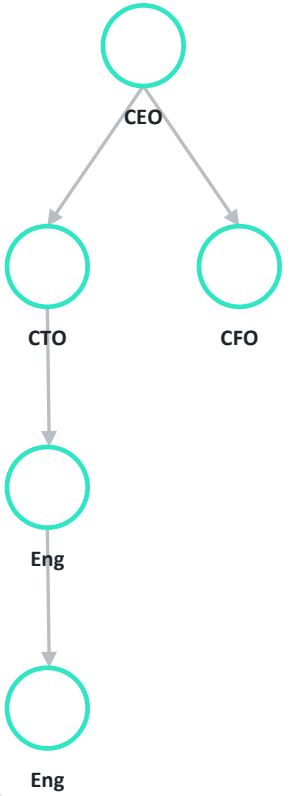

**Step 2: + CTO, CFO**




**Step 3: + Eng Lead**



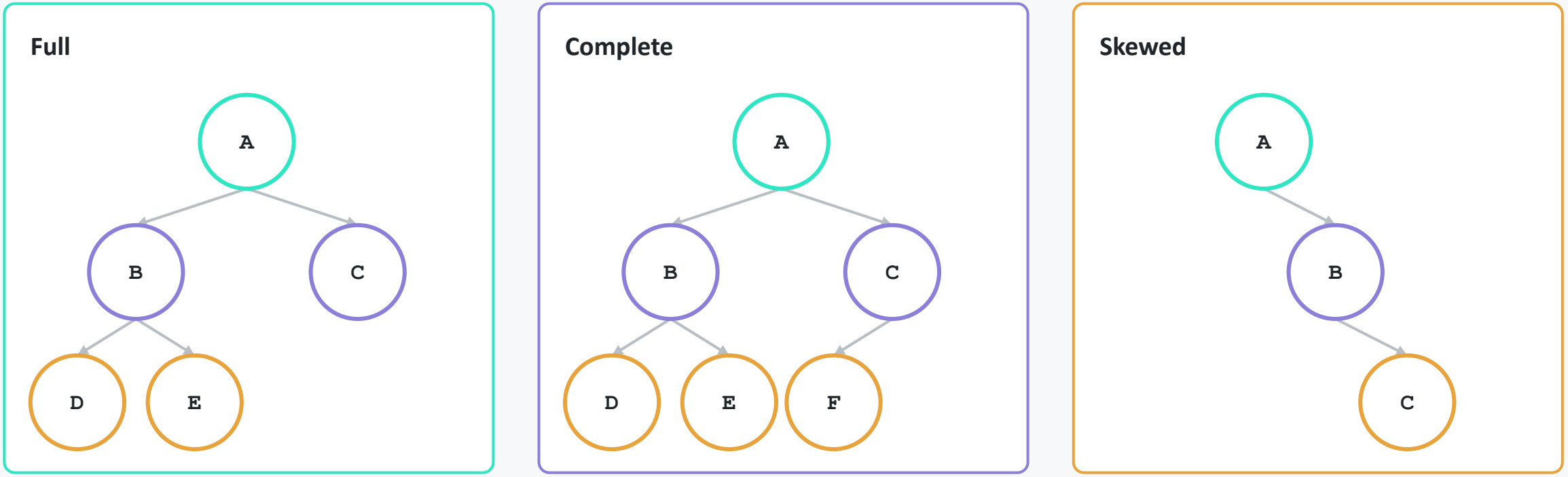

**Step 4: + Engineer**




*Height after Step 4: 3 edges from root (CEO) to deepest leaf (Engineer).*

THREE SHAPES

# Full, Complete, and Skewed Binary Trees



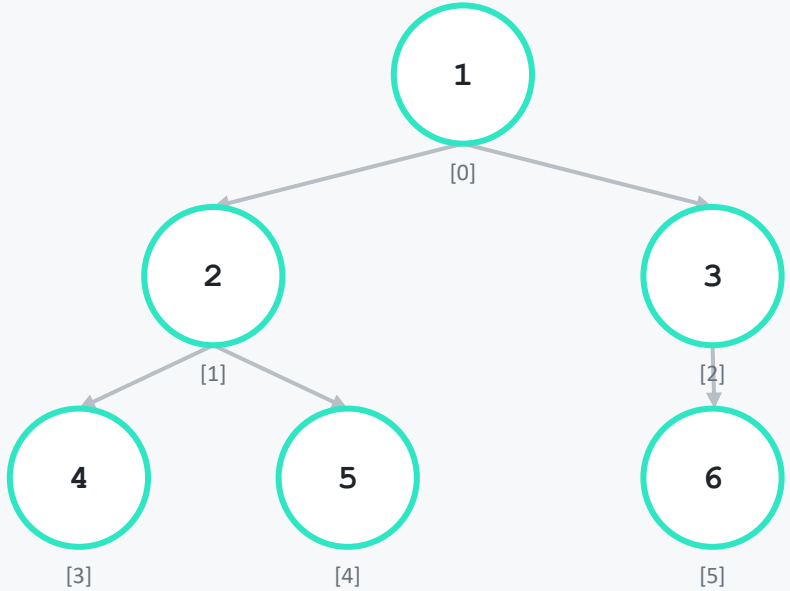

Full: every node has 0 or 2 children. Complete: every level full except possibly the last, left to right. Skewed: every node has one child — the worst case for search (Lecture 21).

FOR COMPLETE TREES SPECIFICALLY

# Sequential (Array) Representation

**index arithmetic**

```
left  = 2*i + 1
right = 2*i + 2
parent = (i - 1) / 2
```





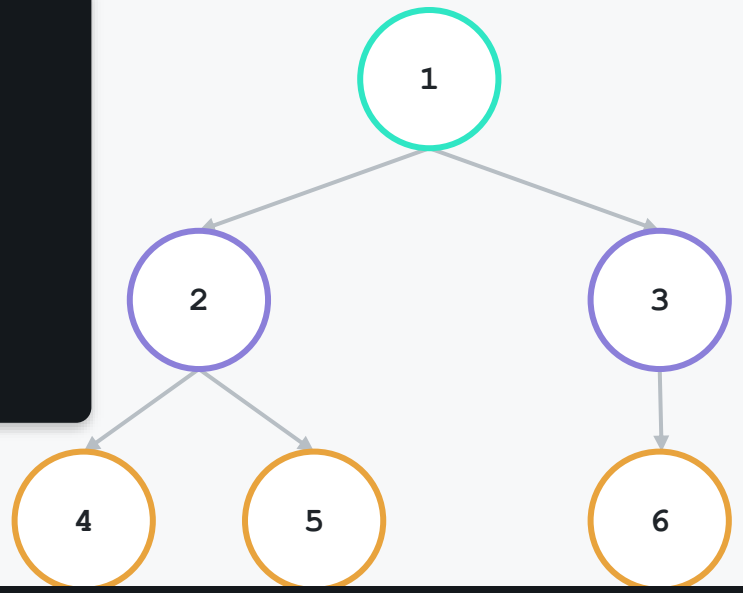
*Compact and cache-friendly — but wastes space badly for a skewed (incomplete) tree.*

ANY SHAPE, NO WASTE

# Linked Representation of Binary Trees

**TreeNode**

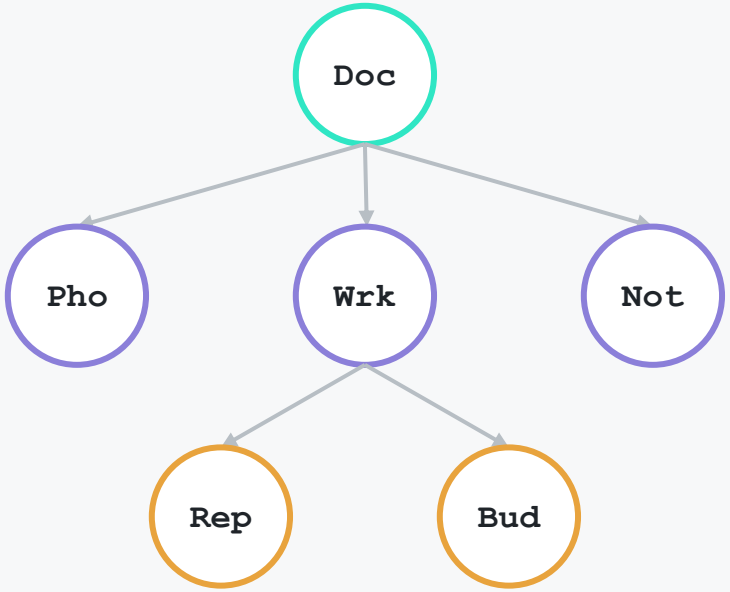
```
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int v) : data(v),
        left(nullptr), right(nullptr) {}
};
```



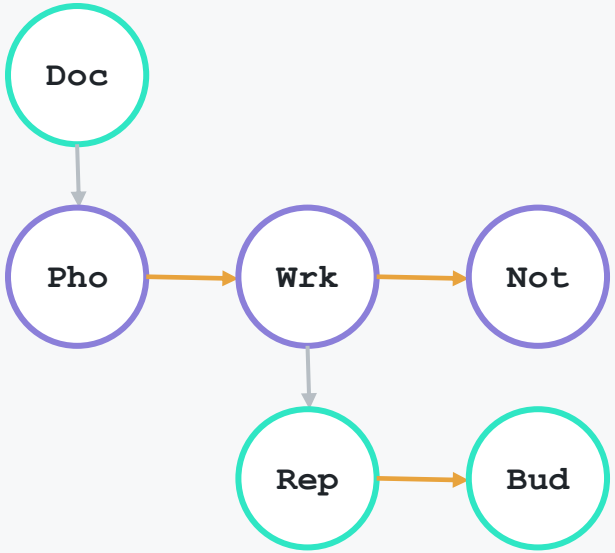

| | Array Representation | Linked Representation |
|---|---|---|
| **Best for** | Complete trees (heaps) | Any binary tree shape |
| **Memory** | Compact, wastes on gaps | One node + 2 pointers each |
| **Finding a child** | O(1) arithmetic | O(1) pointer dereference |

# First-Child / Next-Sibling (FCNS)

```
left = this node's FIRST CHILD.    right = this node's NEXT SIBLING.
```




General tree (Documents: 3 children)




FCNS binary encoding (down=firstChild, right=nextSibling)

LOSSLESS, BY CONSTRUCTION

# The FCNS Node and Conversion

general_to_binary.cpp

```
struct FCNSNode {
    string name;
    FCNSNode* firstChild;   // was "left"
    FCNSNode* nextSibling;  // was "right"
};

FCNSNode* convert(GeneralNode* node) {
    FCNSNode* converted = new FCNSNode(node->name);
    converted->firstChild = convert(node->children[0]);
    FCNSNode* sibling = converted->firstChild;
    for (size_t i = 1; i < node->children.size(); i++) {
        sibling->nextSibling = convert(node->children[i]);
        sibling = sibling->nextSibling;
    }
    return converted;
}
```

WHERE TREES SHOW UP

# Real-World Applications of Trees

### File Systems

A folder can hold any number of items — navigation is naturally hierarchical.

### Org Charts

A manager's "next" isn't one person — it's a whole team of direct reports.

### The DOM

A  can contain any number of children — rendering & events walk this tree.

### Decision Trees

Each answer branches to a completely different next question.

### Syntax Trees

a + b * c nests b * c inside the addition — the structure IS a tree.

KEY TAKEAWAYS

# Key Takeaways

- A tree is a hierarchical, non-linear structure — one node can have multiple children, unlike every linear structure so far.
- Core terminology — root, parent, child, sibling, leaf, depth, height, subtree — is used throughout the rest of this unit.
- A binary tree restricts every node to at most two children; it can be full, complete, or skewed, which affects search efficiency.
- The array representation is compact but only efficient for complete trees; the linked representation handles any shape.
- A general tree can be represented with ordinary two-pointer nodes via first-child/next-sibling (FCNS) encoding.