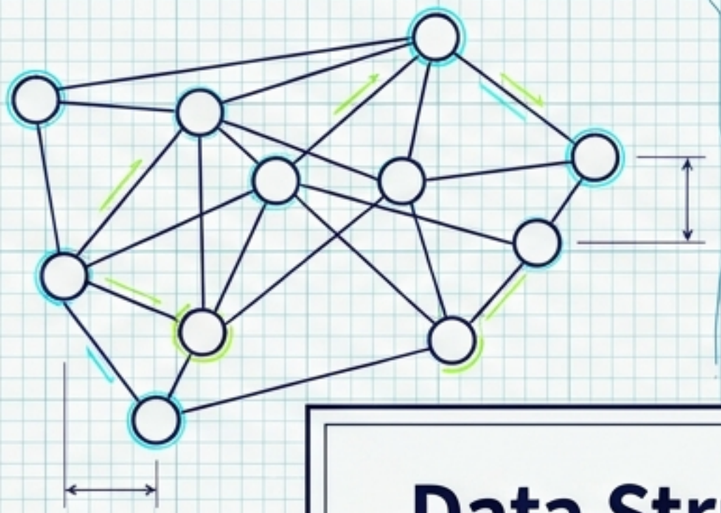
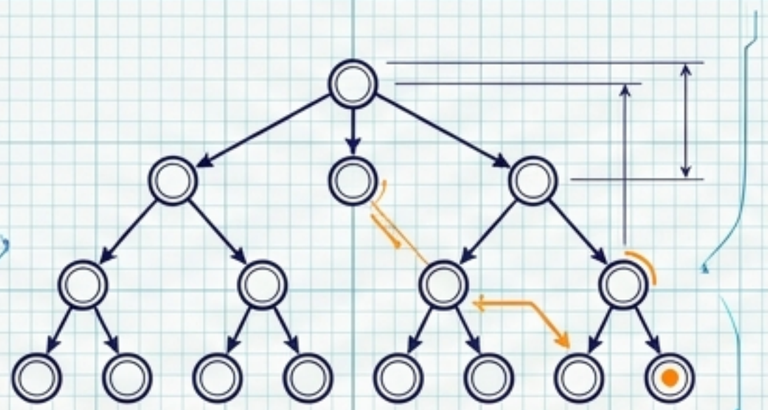
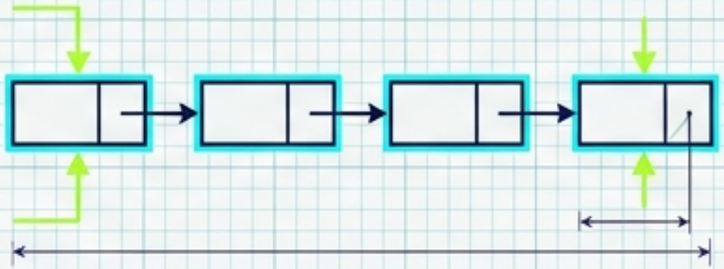






# Data Structures: 從零到精通的架構藍圖

## Data Structures: The Architect's Blueprint to Mastery

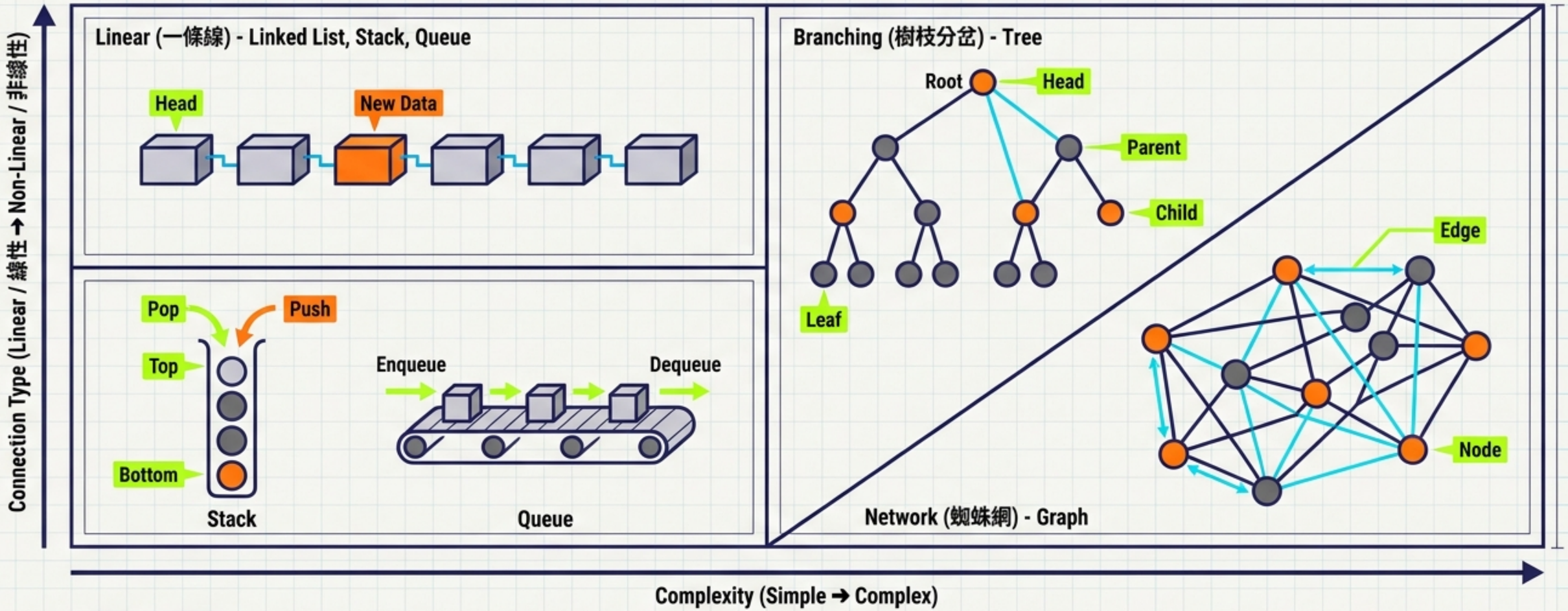
核心數據結構完全解析：LL / Stack / Queue / Tree / Graph

**Professor's Note**

**「這不是一堂死背程式碼的課。這是一場教你如何像架構師一樣思考的訓練。我們將用費曼學習法 (Feynman Technique)，帶你把最抽象的結構變成最直觀的常識。」**

This is not a class on rote memorization. This is training to think like an architect. We will use the Feynman Technique to turn abstract structures into intuitive common sense.

# The Map of the Territory: Evolution of Data Connections




**Professor's Note**

「所有的數據結構，都只是在規定『資料如何連接』的規則。」
All data structures are simply rules dictating how data connects.

- **Linear Structures (線性結構): One-to-one relationship.**
- **Non-linear Structures (非線性結構): One-to-many or Many-to-many relationships.**

# OOP Fundamentals: Interface vs. Class (觸類旁通)

## Interface (介面)


- 定義了「能做什麼」(Defines WHAT can be done).
- 包含 Abstract (抽象) 方法，例如 push(), pop()。

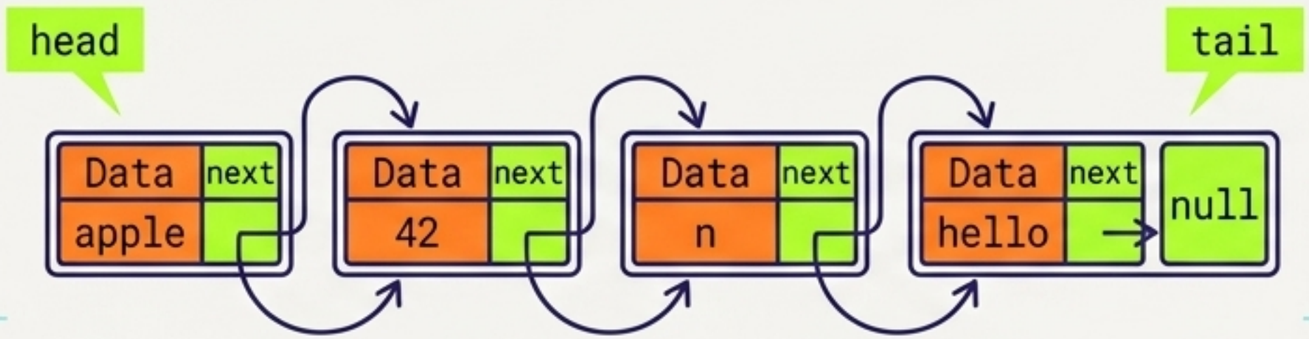
## Class (類別) / Implements (實作)


- 定義了「如何做到」(Defines HOW it is done).
- 例如 LinkedStack 或是 ArrayStack 都是實作同一個 Stack 介面。

**Professor's Insight**

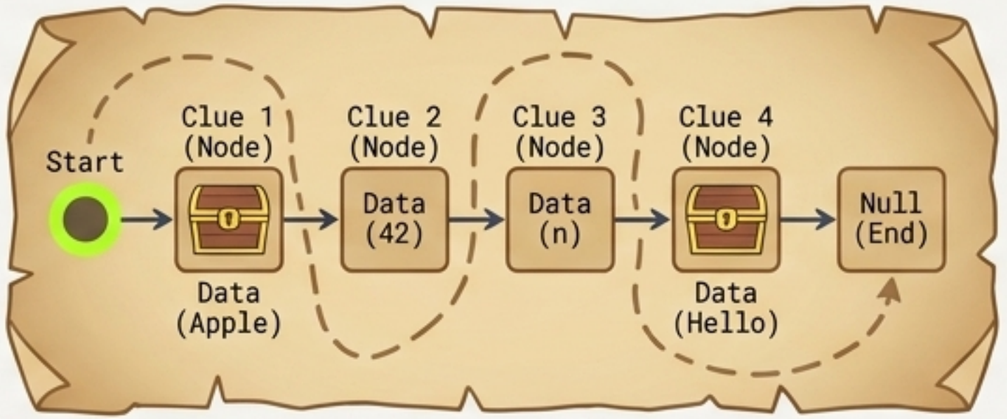
「考試卷常出現 public interface Stack 和 public class LinkedStack implements Stack。為什麼？因為『介面』是合約，『類別』是真正的工人。掌握這點，你就能讀懂所有原始碼。」
Interfaces are contracts; Classes are the actual workers.

# Linked Lists: The Dynamic Chain





## 尋寶遊戲 (A Treasure Hunt)




陣列是固定的置物櫃；而 Linked List 是尋寶遊戲，每一個提示（Node）都包含寶藏（Data）和下一個提示的藏寶圖（Next pointer）。

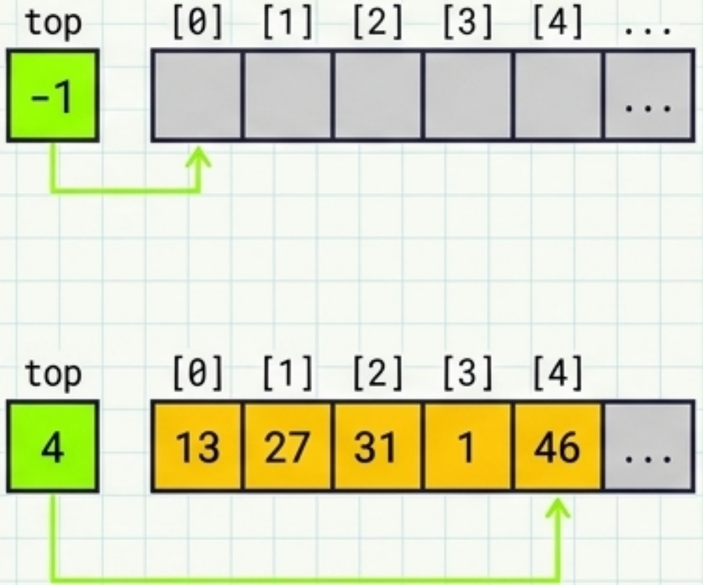
- **Node** (節點): Contains **Data** and **Next** reference.
- **Head** (頭部): The starting point.
- **Tail** (尾部): Points to the last node. (Saves O(N) time when adding to the end!)
- **Null** (空值): Marks the end of the line.

# Stacks: Constraining Access with LIFO

## Feynman Analogy


**Core Rule: LIFO**
(Last-In-First-Out / 後進先出)

## Technical Implementation

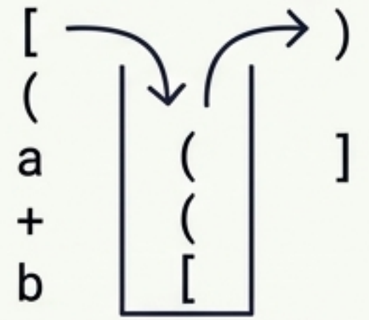



## Primary Operations & Application

- **push(item)**: 將資料壓入頂部 (Add to top).
- **pop()**: 移除並回傳頂部資料 (Remove and return top).
- **top()**: 僅查看不移除 (View top without removing).

**Practical Exam Application**

Application: 括號檢查 (Parenthesis matching) [(a+b)]

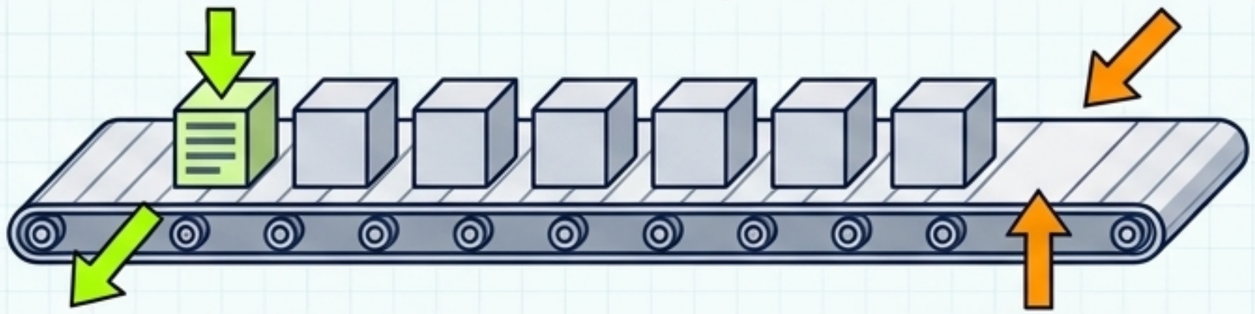


# **Queues:** Fairness in Data with FIFO

## Mechanics Analysis

**front: 指向第一個元素**
(Points to the first element)

• **enqueue**(item) ➡ enters at rear

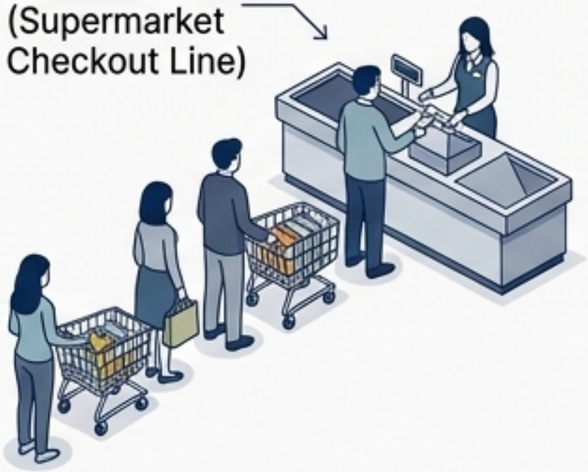


• **dequeue()** ➡ exits at front

**rear: 指向下一個空位**
(Points to the next available position)

## Feynman Analogy

**超市排隊結帳**
(Supermarket Checkout Line)

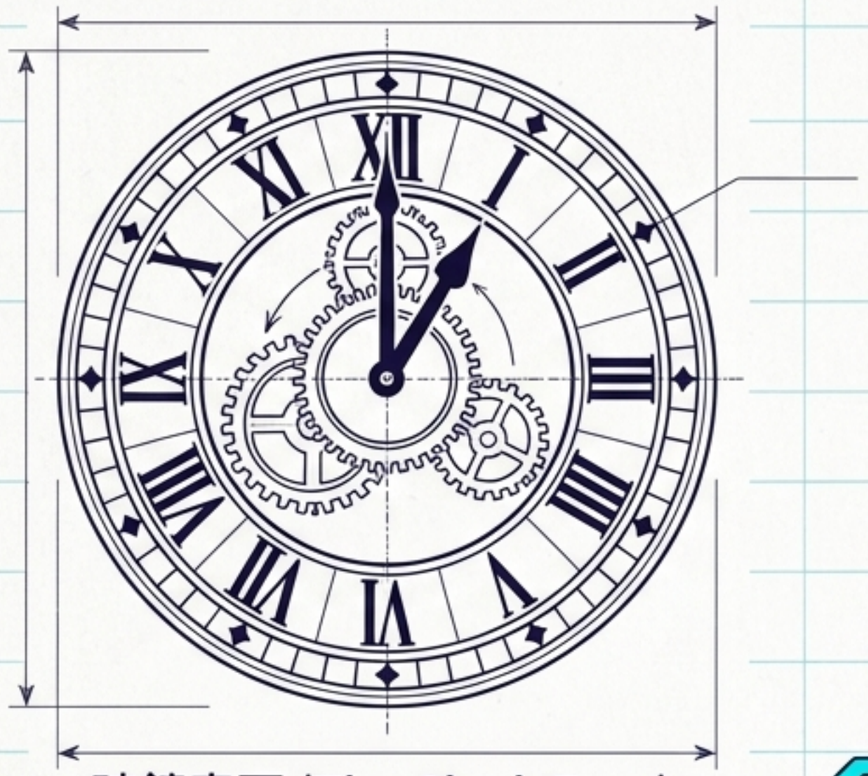


**Core Rule: FIFO**
(First-In-First-Out / 先進先出)

## ⚠ Architectural Flaw: The Memory Leak

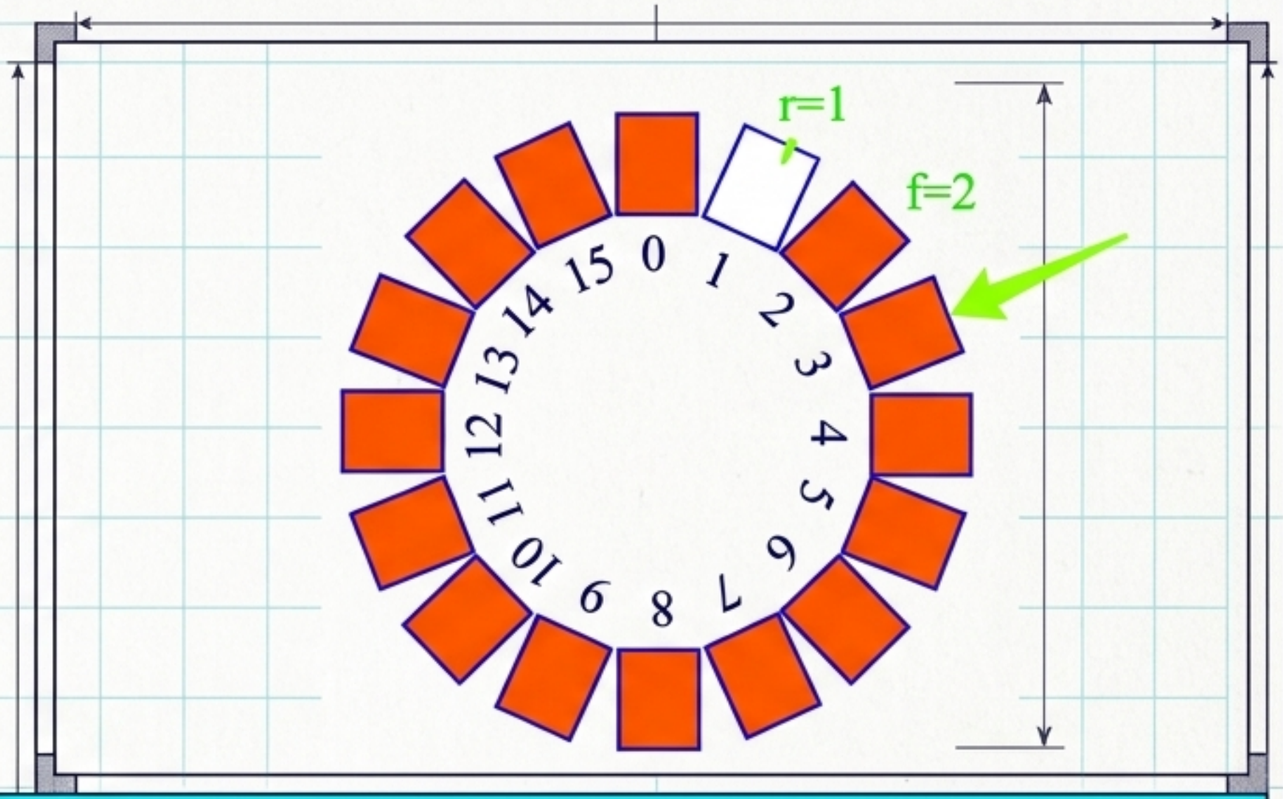

When implemented with arrays, moving pointers rightward creates a 'fake full' error.
The array hits the wall even if the front is empty!

# The Circular Queue: Elegant Math Solutions



時鐘表面 (The Clock Face)

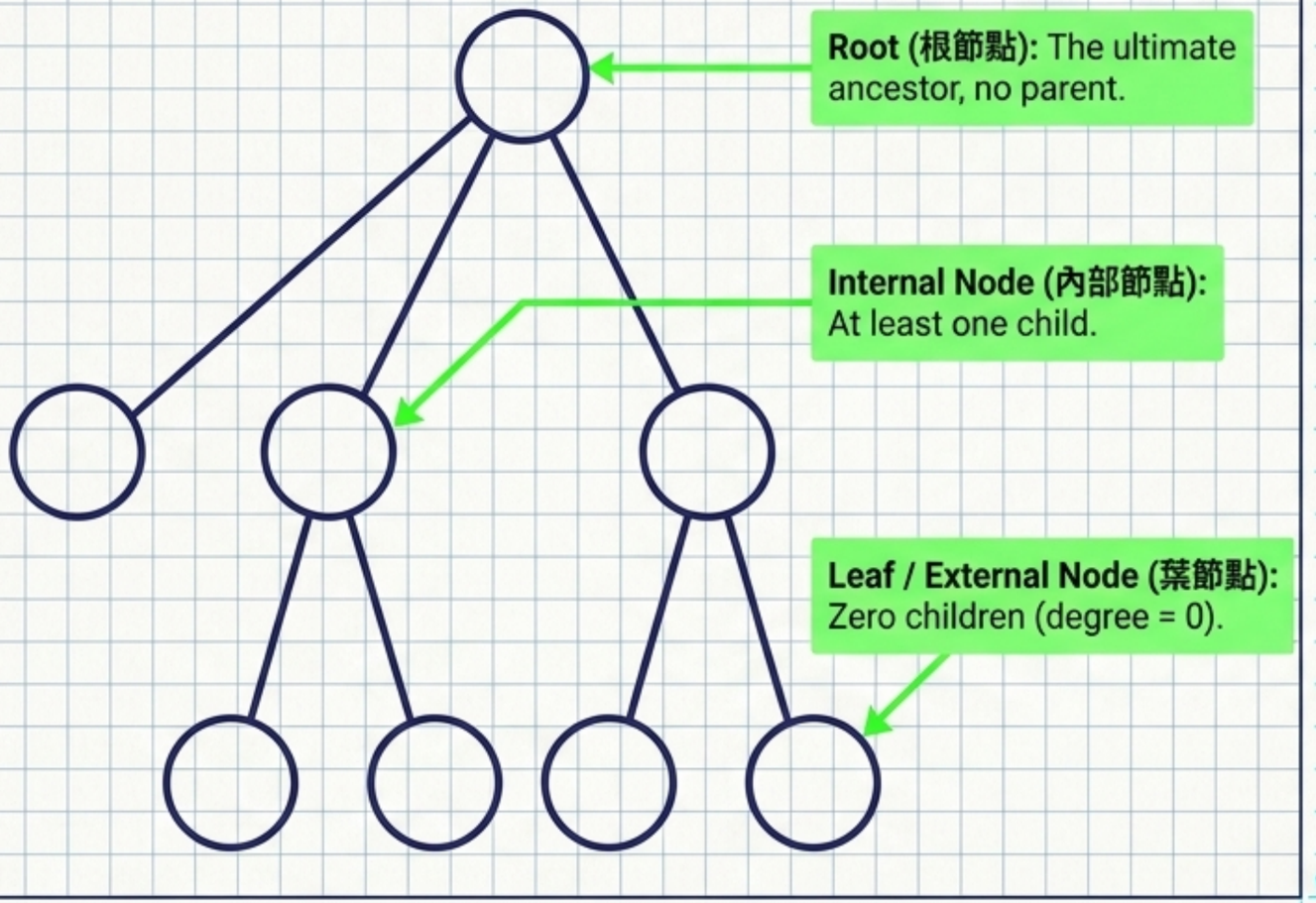
「12點加1小時是1點，不是13點。這就是 **Modulo (%)** 的魔力！」
(12 o'clock plus 1 hour is 1 o'clock, not 13. This is the magic of Modulo!)



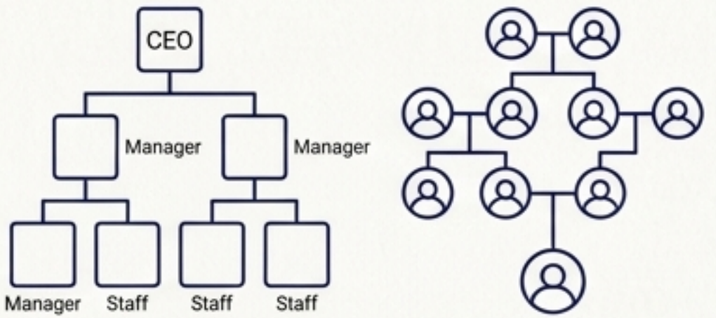

## Crucial Formulas (Must Memorize)

- **Move Rear: rear = (rear + 1) % capacity**
- **Move Front: front = (front + 1) % capacity**
- **Size Calculation: (N – front + rear) % N**

# Trees: Breaking Linear Constraints




## Feynman Analogy

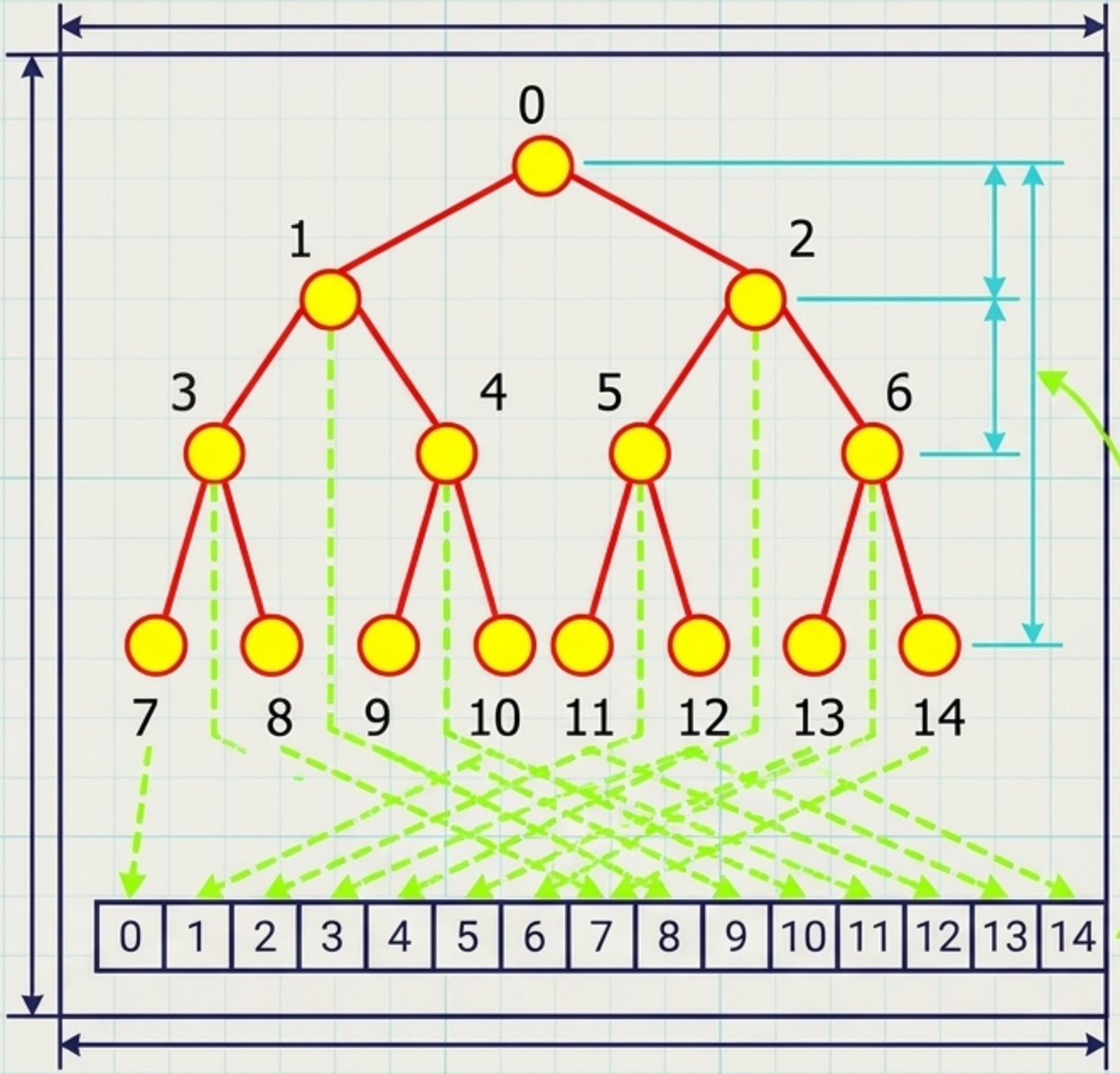



家族族譜 (Family Tree) /
公司組織架構 (Corporate Hierarchy)

## Definition

- **Height** (高度): The longest path from root to a leaf.
- **Degree** (分支度): Number of children a node has.

# Binary Trees: The Rule of Two & Array Magic




## Professor's Insight & Formulas

「當我們規定一棵樹『最多只能有兩個小孩』時，奇蹟發生了。我們甚至不需要指標（**Pointers**），可以直接用數學將樹存入一維陣列中！」
(When we restrict nodes to max 2 children, we don’t need pointers; we map it flat using math!)

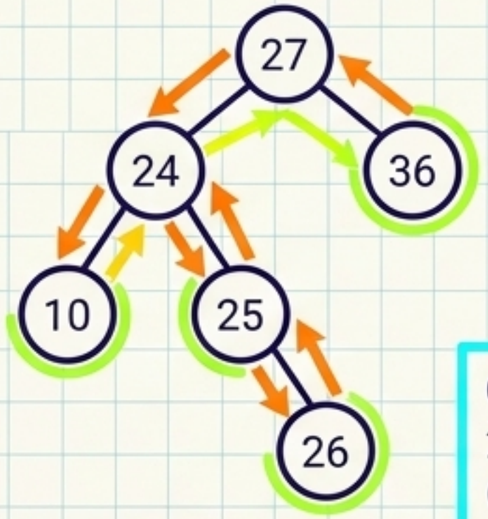
**Array Mapping Formulas:**

- Parent at index **p** (父節點位置)
- Left Child (左子節點): **2p + 1**
- Right Child (右子節點): **2p + 2**

# Tree Traversals: Reading a 2D Map in 1D

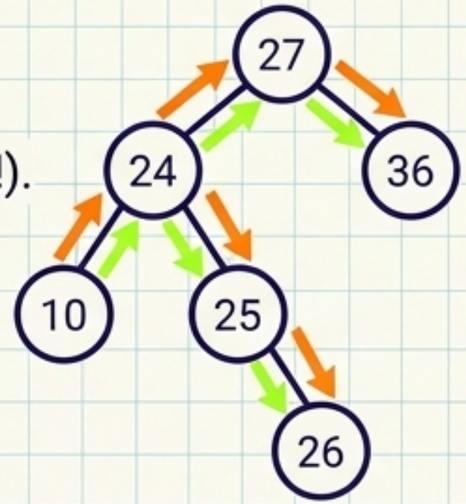
## Pre-order (前序)

Root ➔ Left ➔ Right.
(Top-down scanning).




## In-order (中序)

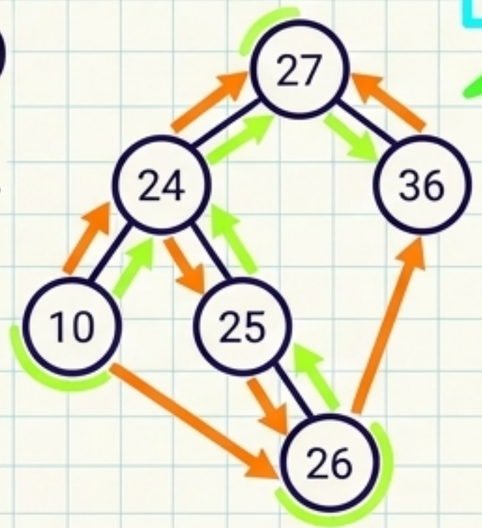
Left ➔ Root ➔ Right.
(Produces sorted output!).




**Core Concept:**
決定於何時拜訪「父節點」
(It all depends on when you visit the Parent/Root).

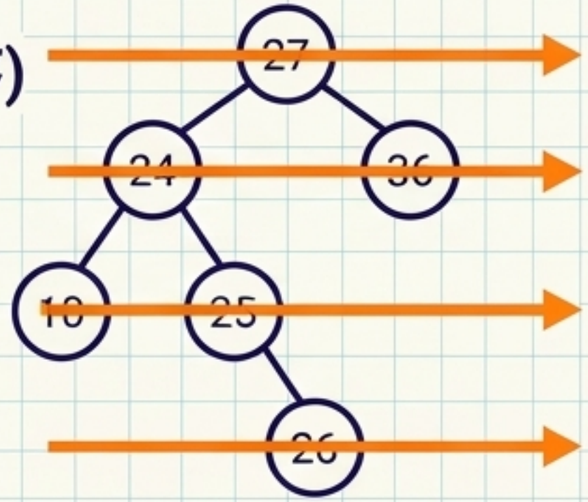
## Post-order (後序)

Left ➔ Right ➔ Root.
(Bottom-up scanning).




## Breadth-first (層序)

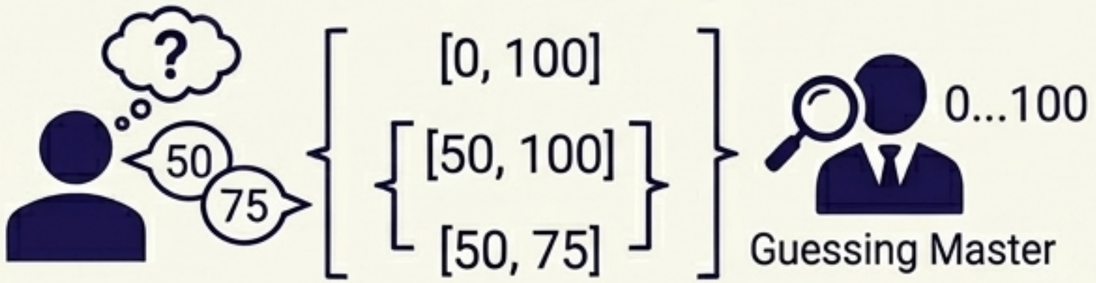
Level by level, 0 to N.



# Binary Search Tree (BST) & AVL: Sorting for Speed

## Feynman Analogy & Concept

### The Number Guessing Game

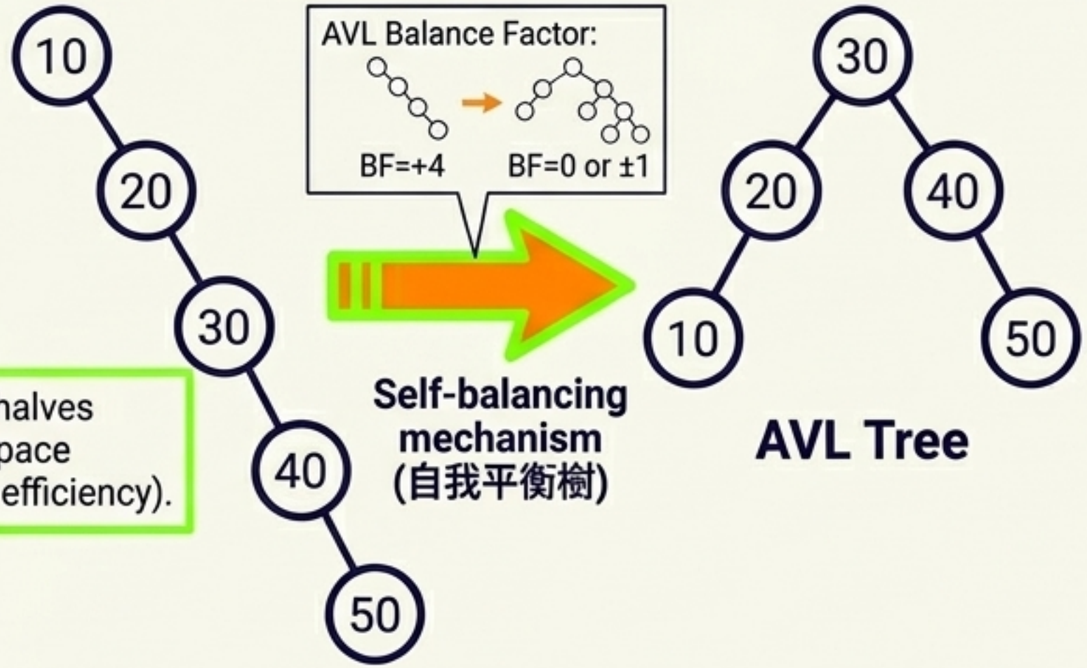



終極猜數字遊戲
(The Ultimate Number Guessing Game)

**Left < Parent < Right**

Each guess halves the search space (logarithmic efficiency).

## The Balance Problem & AVL Solution




## Performance Matrix - Exam Focus

| Structure | Best Case | Worst Case |
|---|---|---|
| BST | O(1) (Target is Root) | Height + 1 (Degrades to Linked List) |
| AVL Tree | O(1) | Logarithmic O(log N) |

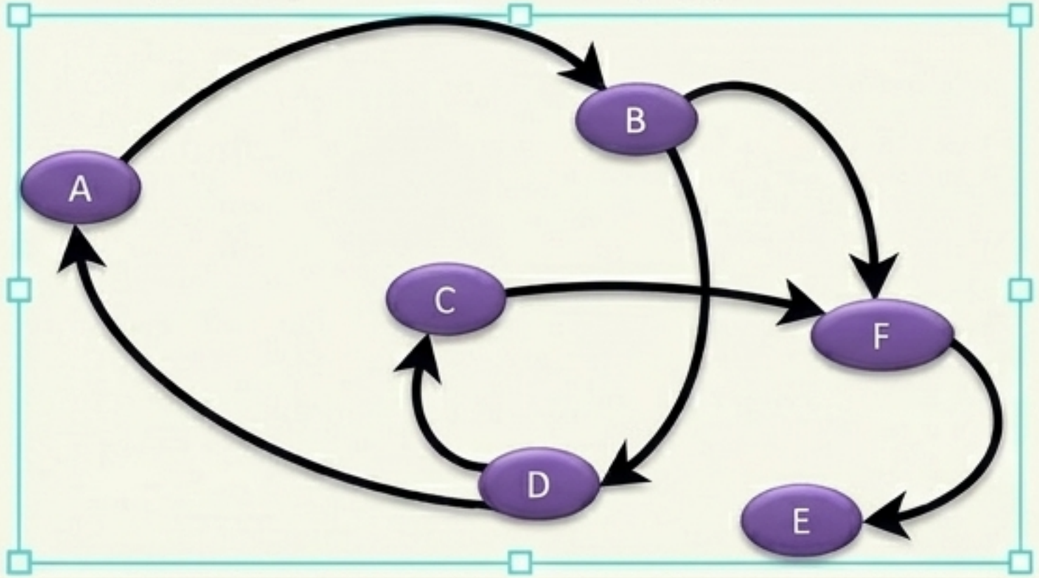
AVL ensures worst-case remains logarithmic, never linear.
(The cost of speed is extra space for balance factors.)

# Graphs: The Ultimate Data Structure Freedom

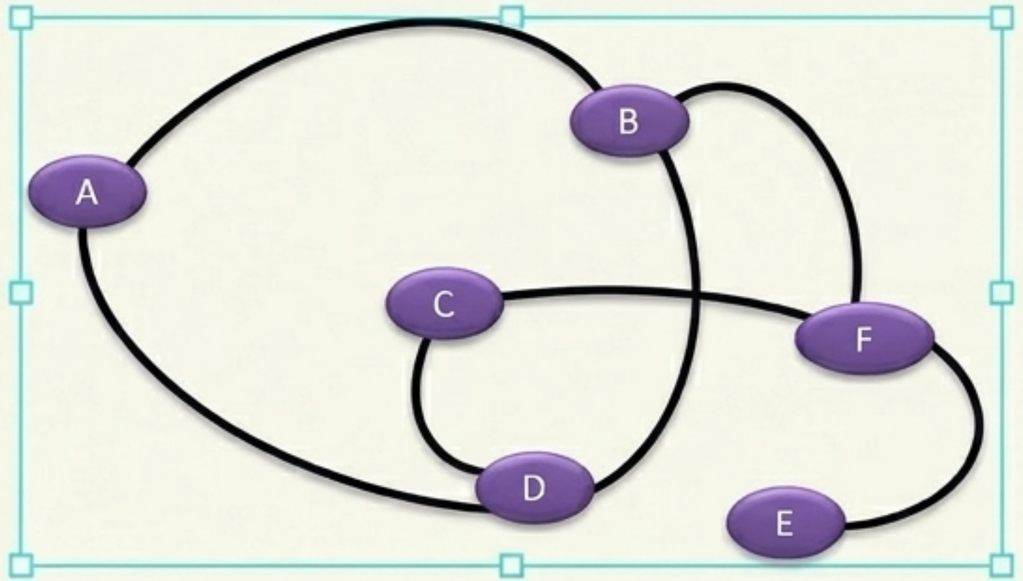
## Feynman Analogy & Concept

捷運路線圖 (MRT System Map) / 社群網路 (Social Networks)

**Directed (有向圖):** One-way streets (e.g., Twitter followers).




**Undirected (無向圖):** Two-way streets (e.g., Facebook friends).



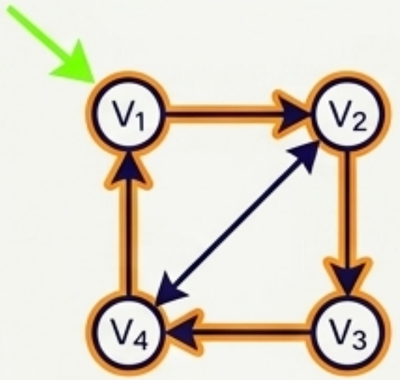

## Vocabulary Matrix - Exam Focus

- **Vertex / Vertices (頂點, V):** The nodes/stations.
- **Edge (邊, E):** The connections/routes.

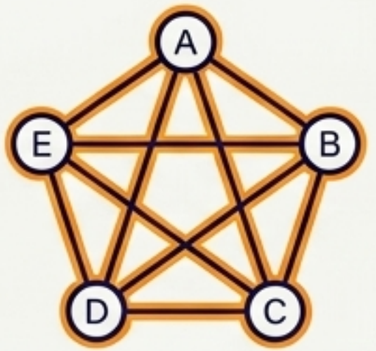
**Professor's Insight:**
「拿掉 Root，拿掉 Parent/Child 限制。
任何節點都能連結任何節點，任何節點都能連結任何節點，這就是圖 (Graph)。」
(Remove root, remove parent rules. Anyone connects to anyone.)

# **Graph Sub-structures:** Navigating Exam Terminology

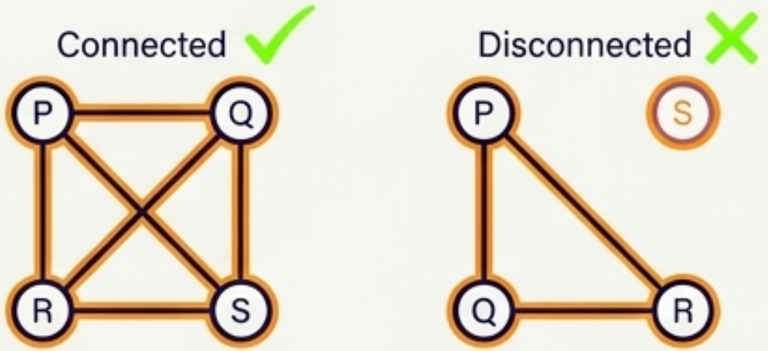
**Cycle (循環):** A path starting and ending at the same vertex.



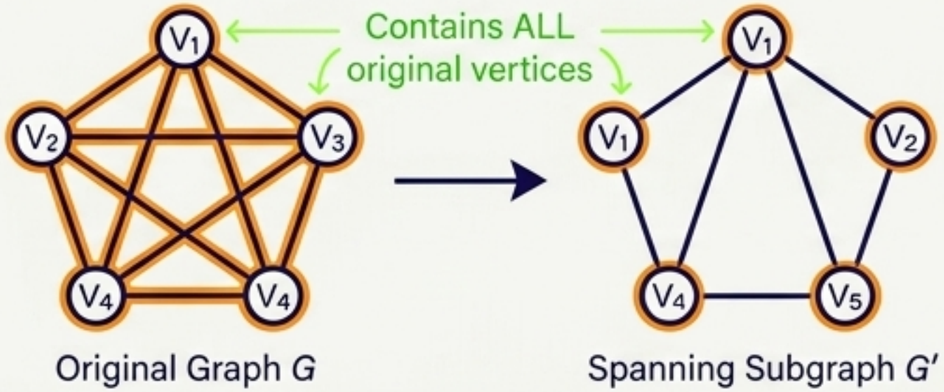

**Complete Graph (完全圖):** Everyone is directly connected to everyone. Formula: $|V| \times (|V| - 1) / 2$




**Connected Graph (連通圖):** A path exists between any two distinct vertices.

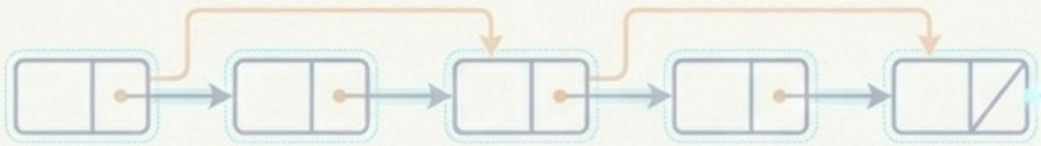


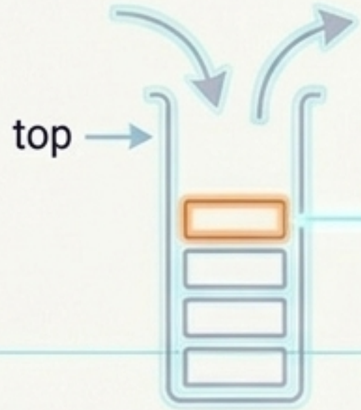

**Spanning Subgraph (生成子圖):** Contains ALL original vertices V, but only some edges E.

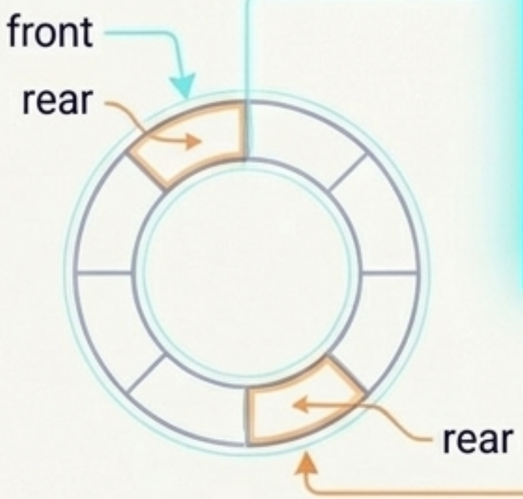


# The Synthesis Matrix: The Master Cheat Sheet

| | Structure (結構) | Core Rule (核心法則) | Feynman Analogy (現實比喻) | Primary Use Case (主要應用) |
|---|---|---|---|---|
| 1 | Linked List  | Dynamic Chain | Treasure Hunt | Memory management |
| 2 | Stack  | LIFO | Pringles Can | Parentheses matching / Reversal |
| 3 | Queue  | FIFO | Checkout Line | OS Scheduling |
| 4 | Tree  | Hierarchy | Family Tree | Searching / Sorting |
| 5 | Graph  | Free Network | MRT Map | Routing / Navigation |






# The Tao of Data (數據之道)

**「世界上沒有『最完美』的數據結構，只有對你的問題來說『最合適』的數據結構。理解它們的代價與優勢，你就能設計出改變世界的架構。」**

There is no ‘perfect’ data structure, only the ‘right’ one for your specific problem. Understand the tradeoffs, and you can design architectures that change the world.

**Read. Analyze. Build. (閱讀。分析。建構。)**