Data Structures

TREES (CHAPTER 5)

Tree

Image credit:
https://en.wikipedia.org/wiki/Tree

Image credit: ChatGPT

Image credit: ChatGPT

Upside Down View

1. Root (one)
2. Branch (two or more)
3. Level/hierarchy (many)

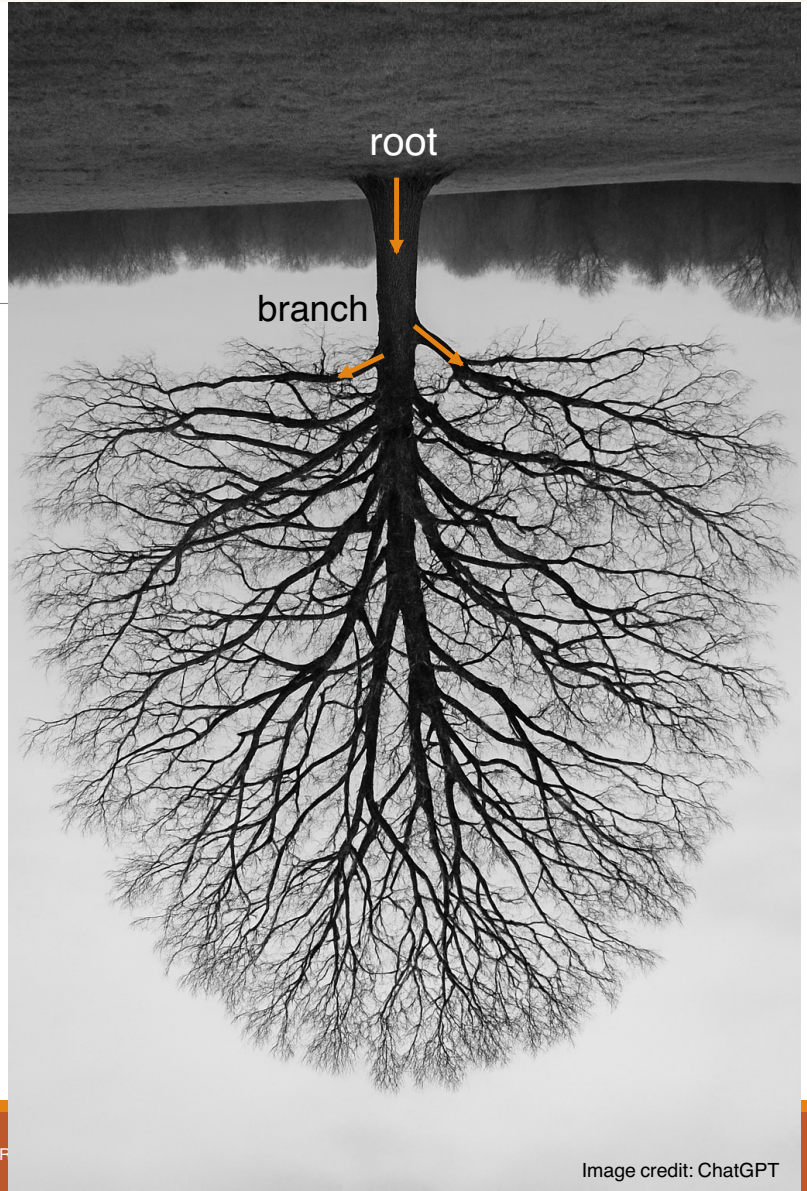


Image credit: ChatGPT

Tree in the Daily Life

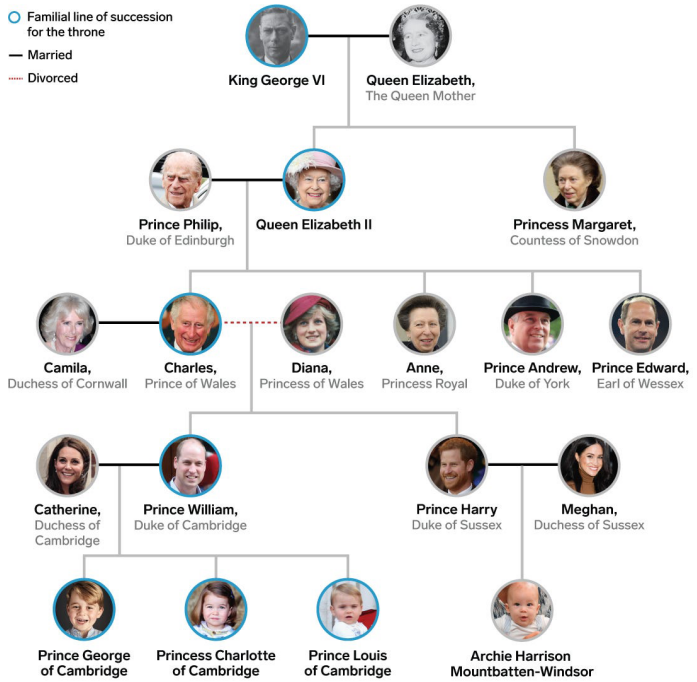
1. Family tree
2. File system hierarchy

File system hierarchy

```
yfhuang@Yu-FengdeMacBook-Pro WorkShop % tree -L 2
.
├── Academic
│   └── PaperReview
├── Administration
│   └── Application
├── Course
│   ├── DataStructures
│   └── IntroductionToDatabaseSystem
└── Research
    ├── GrantProposal
    └── PaperSubmission

11 directories, 0 files
```

RoyalFamily tree



https://www.businessinsider.com/royal-family-tree-british-monarchy-house-of-windsor-2018-5#king-george-vis-descendants-dominate-the-current-line-of-succession-2

Any Ideas?

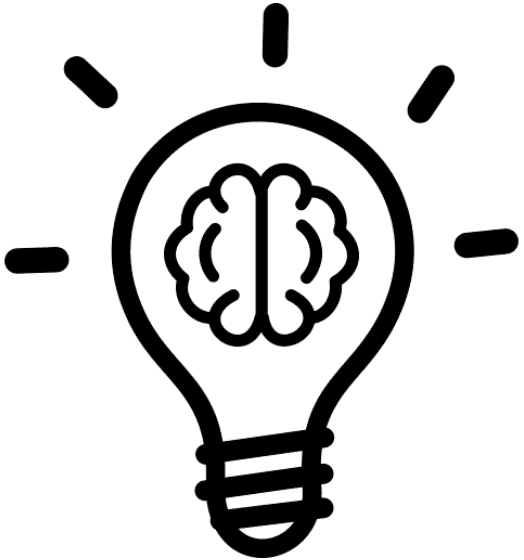


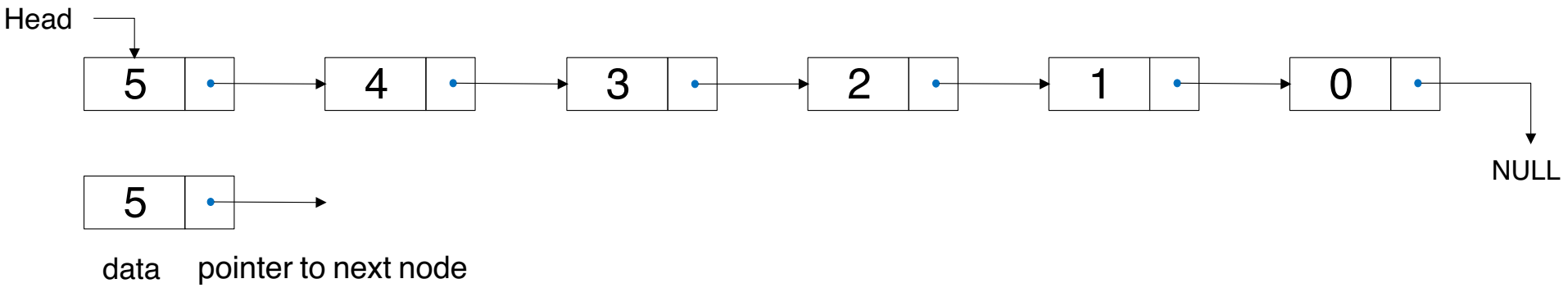
Image credit: https://uxwing.com/idea-icon/

Recap: Linked List

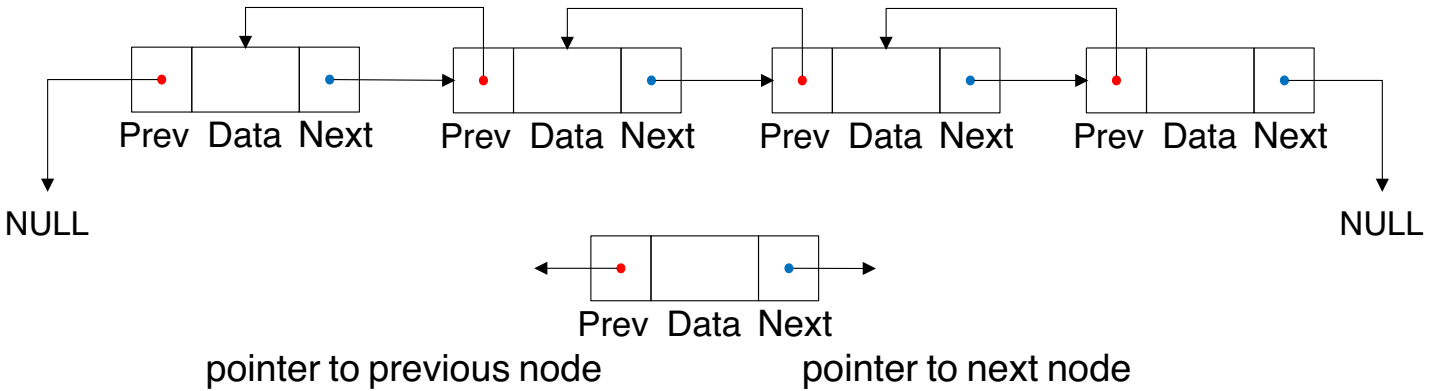
Structure	Pointer(s)	Description	Visualization
Singly Linked List	`next`	Each node connects to one successor. Linear structure.	`A → B → C → NULL`
Doubly Linked List	`prev`, `next`	Each node connects to both predecessor and successor.	`NULL ← A ↔ B ↔ C → NULL`
Binary Tree Node	`left`, `right`	Each node can connect to two children — branching structure.	`A → (B, C)`

If a node in a linked list can have multiple pointers, it can branch — turning a linear structure into a hierarchical one.

Singly Linked List



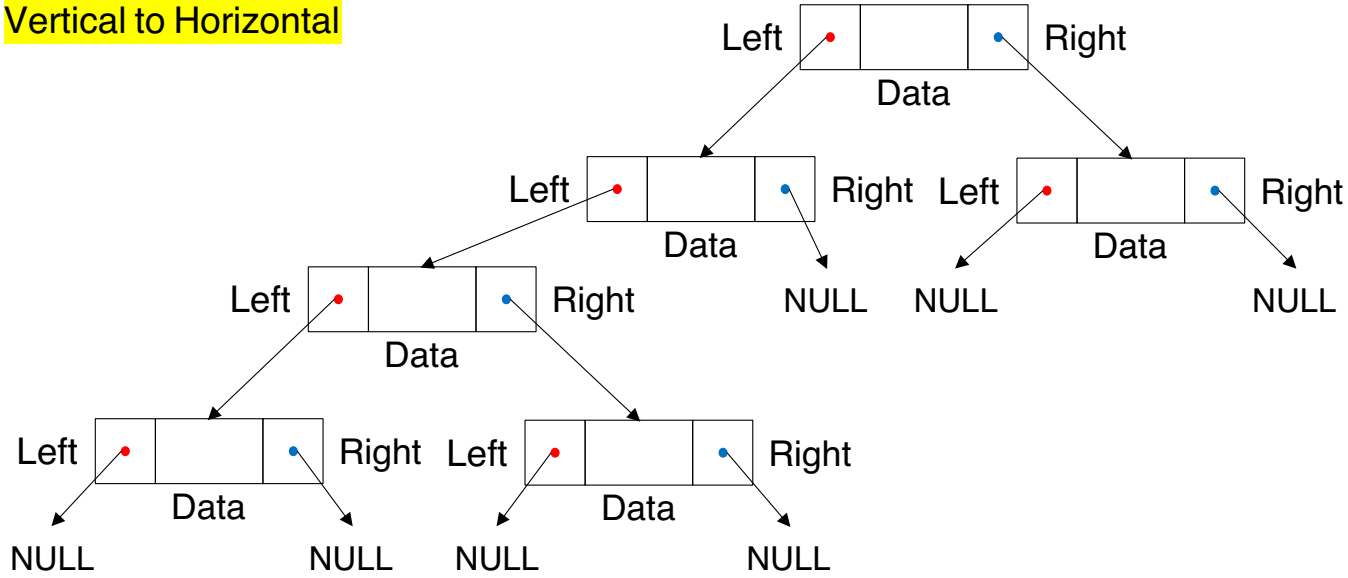
Doubly Linked List



From Linked List to Tree (Linear to Hierarchy)

pointer to previous node pointer to next node

Vertical to Horizontal



Tree in Data Structure

Components

- Node (root, internal, leaf, parent, child, sibling)
- Edge
- Subtree (left, right)
- Level
- Depth (height)
- Branch (fan out)

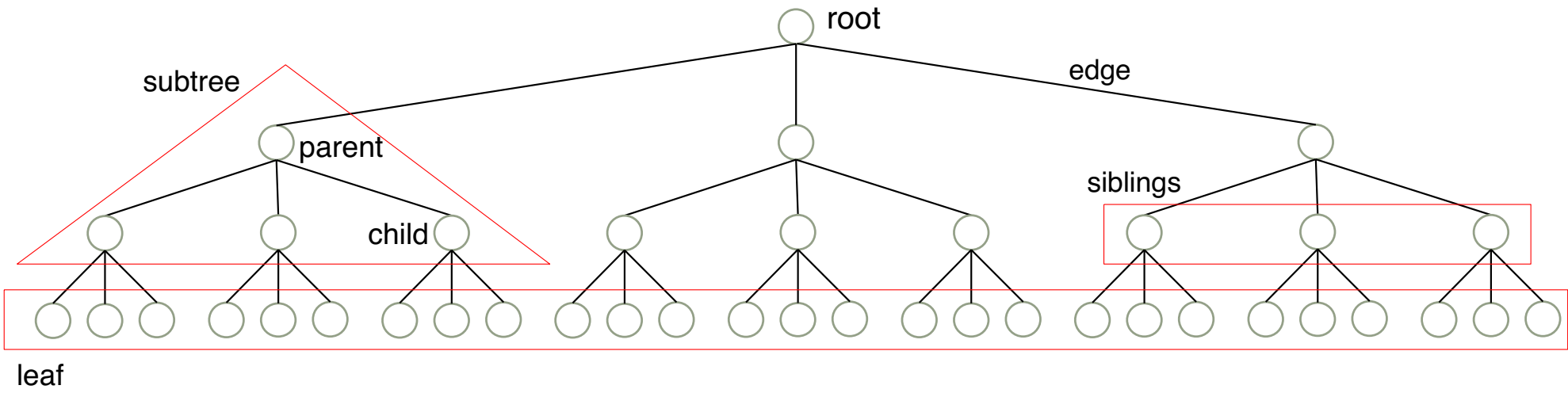
Definition of Tree

A **tree** is a *non-linear hierarchical data structure* consisting of nodes connected by edges.

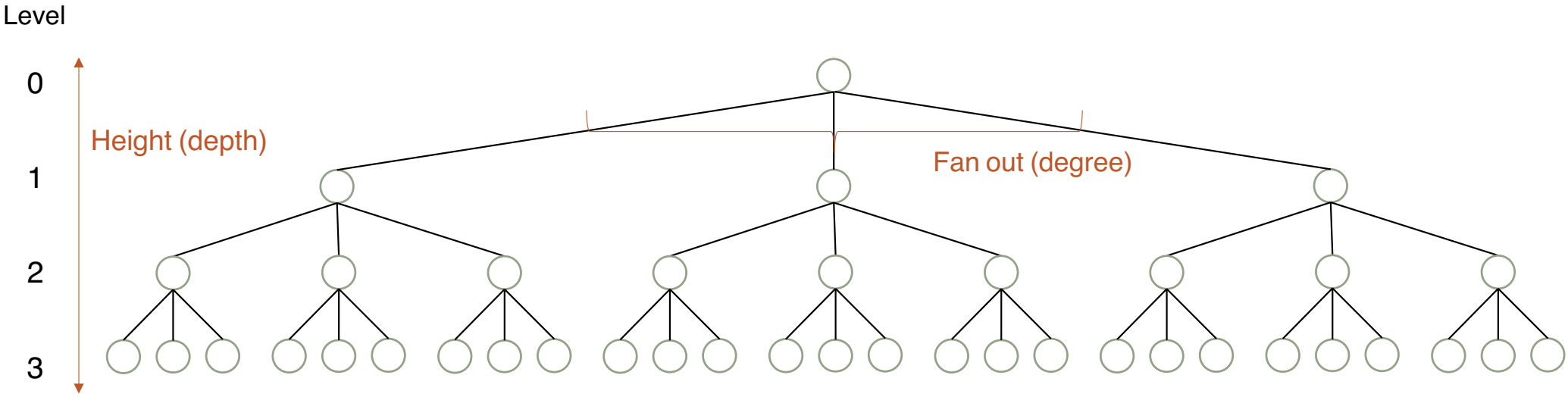
Each node can have zero or more child nodes, forming parent-child relationships.

Trees have no cycles and are commonly used for representing hierarchical relationships like file systems or organizational structures.

Terminology



Terminology



Terminology

Term	Definition
Root	The topmost node of the tree
Parent / Child	A node that has branches leading to other nodes
Leaf	A node with no children
Sibling	Nodes that share the same parent
Edge	A connection between two nodes
Depth / Height	Depth = distance from root; Height = longest path to leaf
Fan-out (Degree)	The number of children a node can have

Types of Trees: Core/Fundamental

Type	Description	Example / Application
Full Binary Tree	Every node has 0 or 2 children	Decision nodes in ML
Complete Binary Tree	All levels filled except possibly the last	Heap (priority queue)
Binary Search Tree (BST)	Left child < parent < right child	Searching, sorting
Balanced Tree (AVL / Red-Black)	Height difference controlled	Efficient search in sets/maps
General Tree	Nodes can have any number of children	Organizational hierarchy, XML/JSON
N-ary Tree	Each node has ≤ N children	Game decision trees (e.g., chess AI)
Trie (Prefix Tree)	Character-based branching, fast string retrieval	Dictionary words, auto-complete
Decision Tree	Nodes represent feature-based decisions	Machine learning classification
Abstract Syntax Tree (AST)	Represents program syntax structure	Compiler design
Spanning Tree	Subset of edges connecting all vertices in a graph	Networking, routing, MST algorithms

Types of Trees: Advanced

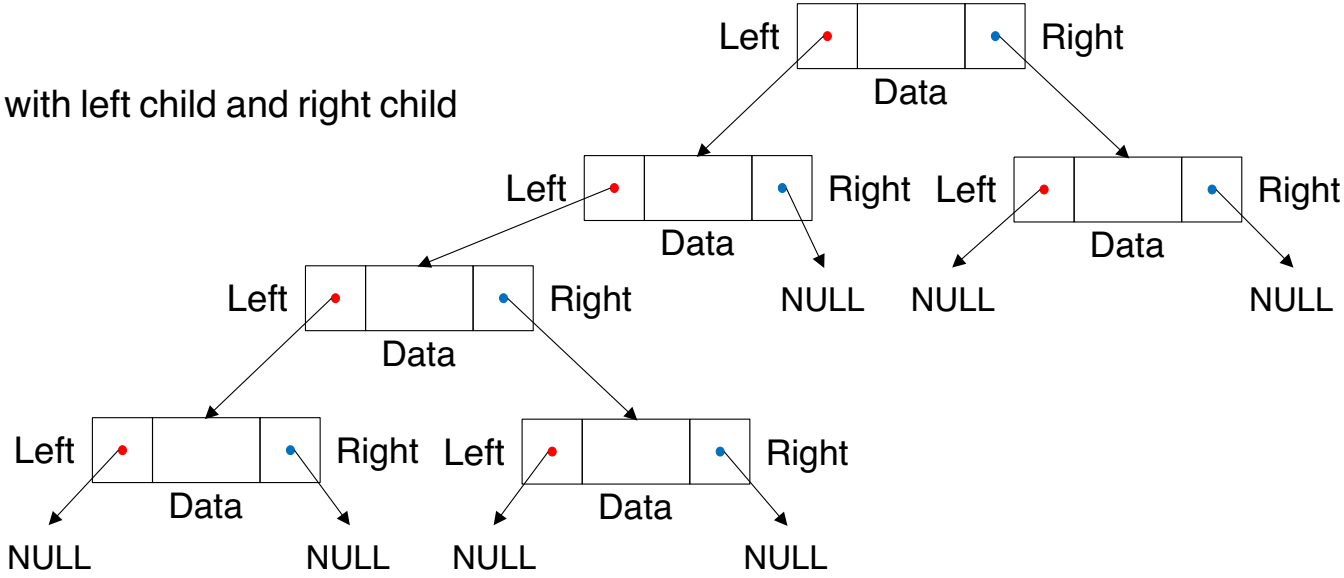
Type	Description	Example / Application
B-Tree / B+ Tree	Multiway search trees designed for disk/block storage	Databases, file systems
Segment Tree	Supports fast range queries (min, max, sum)	Computational geometry, range queries
Fenwick Tree (Binary Indexed Tree)	Efficient prefix sums with log-time updates	Competitive programming
Suffix Tree / Suffix Trie	Efficient pattern and substring matching	String search, bioinformatics
KD-Tree	Space-partitioning for k-dimensional data	Nearest neighbor search (ML)
Quad Tree	Partition 2D space into quadrants	Image compression, GIS
Octree	Partition 3D space into octants	3D graphics, spatial indexing
Heap Tree	Complete tree where parent ≥ or ≤ children	Priority queue
Treap (Tree + Heap)	BST + heap properties combined using random priorities	Balanced randomized BST
Splay Tree	Self-adjusting tree that brings recent nodes to root	Caches, amortized optimization

Tree Traversal

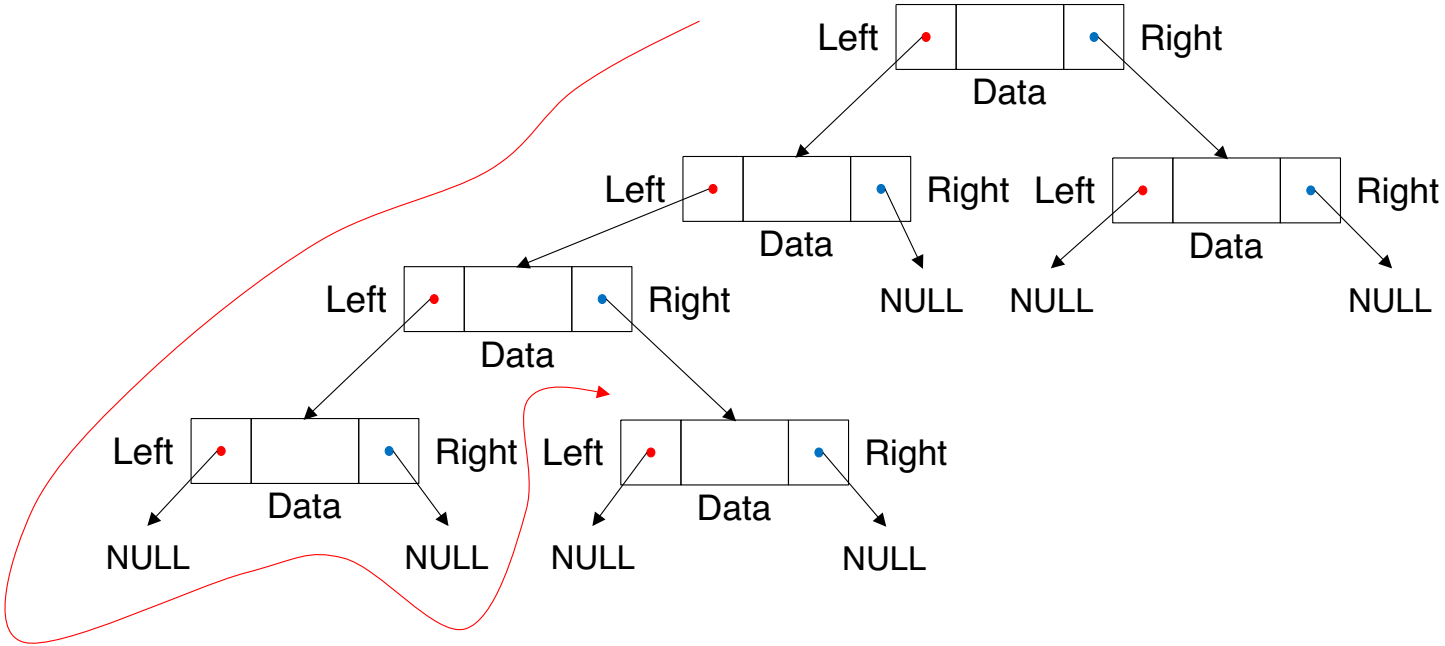
Traversal is the process of visiting each node in a specific order.

- DFT (Depth-First Traversal)
- BFT (Breadth-First Traversal)

walk on the tree node with left child and right child



DFT (Depth-First Traversal)



DFT (Depth-First Traversal)

Order	Visit Sequence	Description
Preorder (Root–Left–Right)	Root first, then children	Used for copying trees
Inorder (Left–Root–Right)	Sorted order for BST	Used in binary search tree
Postorder (Left–Right–Root)	Children first, root last	Used for deletion or freeing memory

Binary Traversal

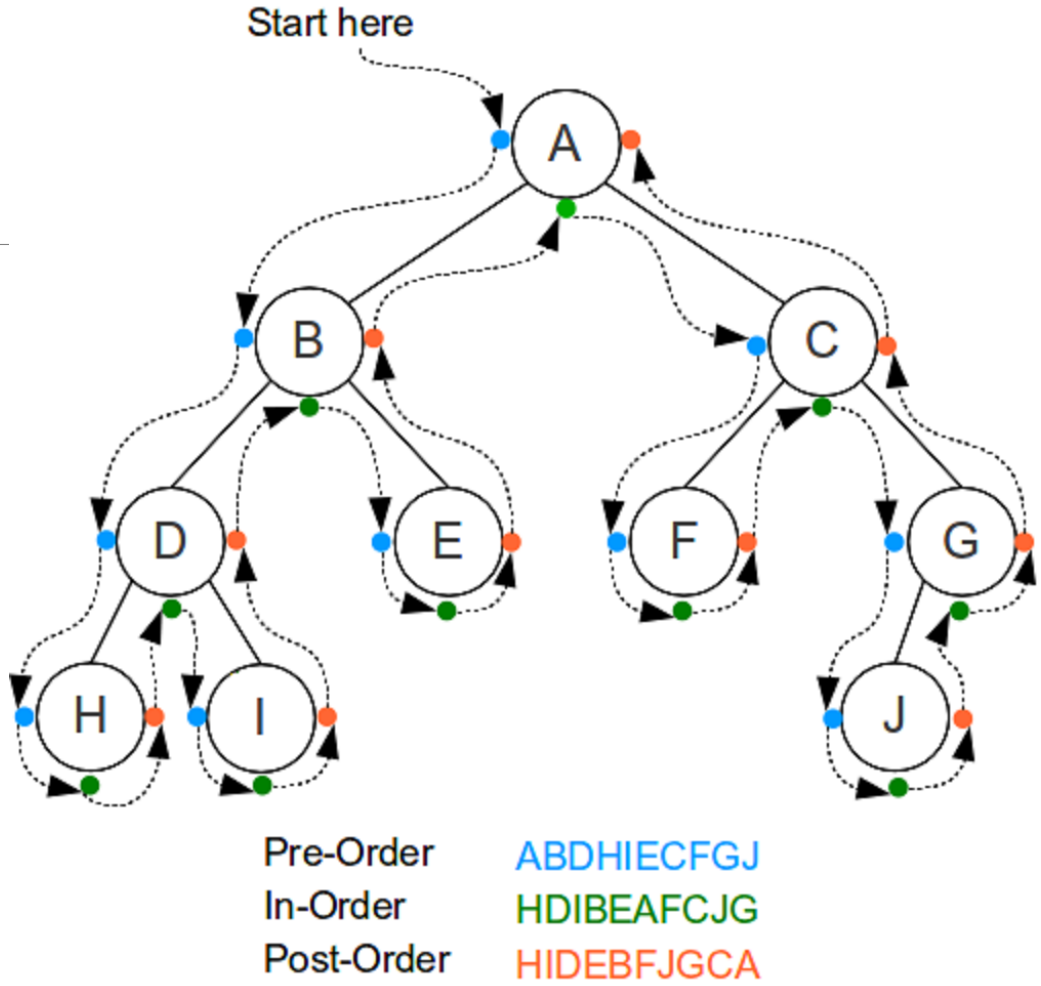
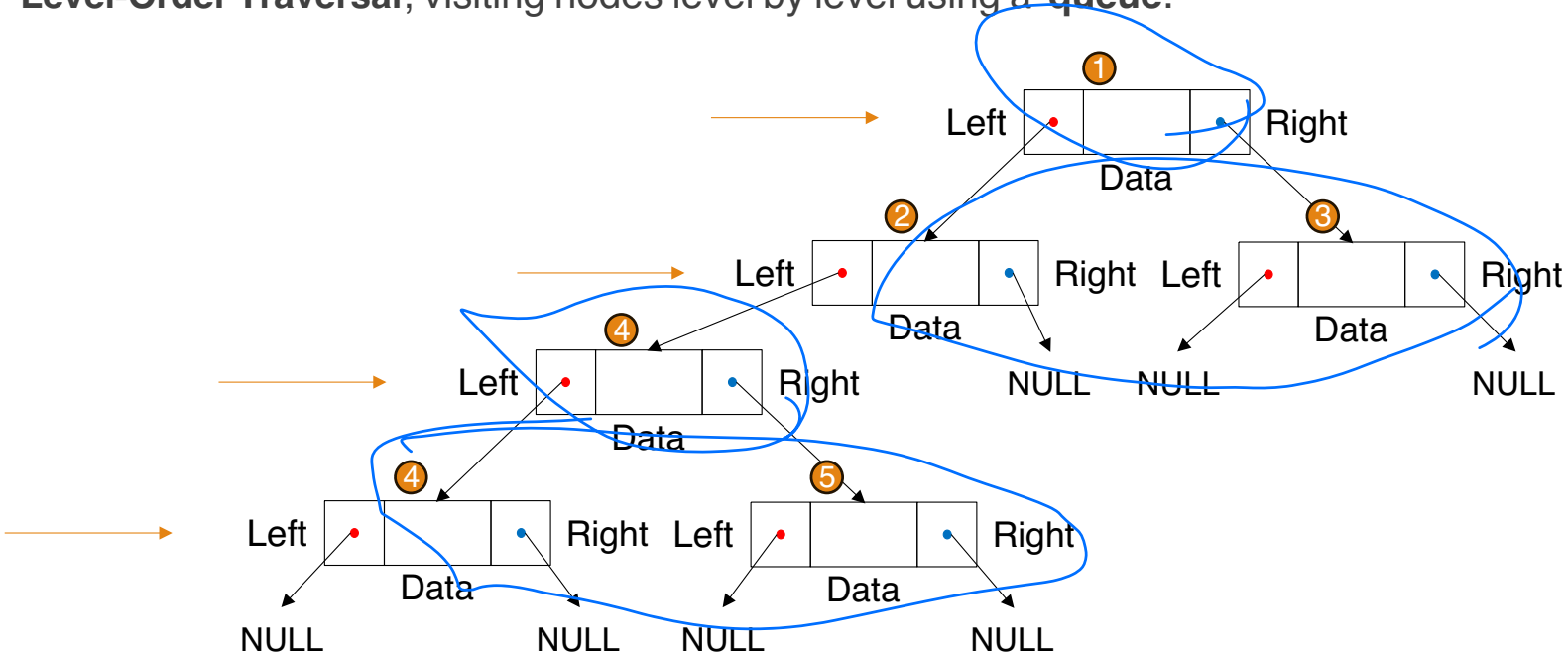


Image credit: https://www.linkedin.com/pulse/binary-trees-representation-traversals-implementation-riya-pandey-1kgff/

BFT (Breadth-First Traversal)

Level-Order Traversal, visiting nodes level by level using a **queue**.



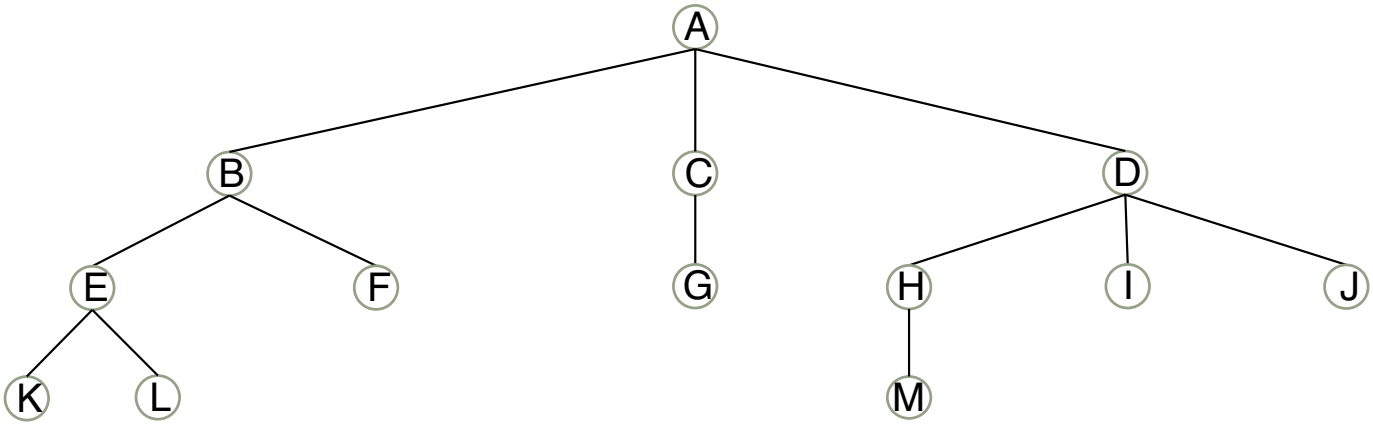
Representation of Tree

List representation

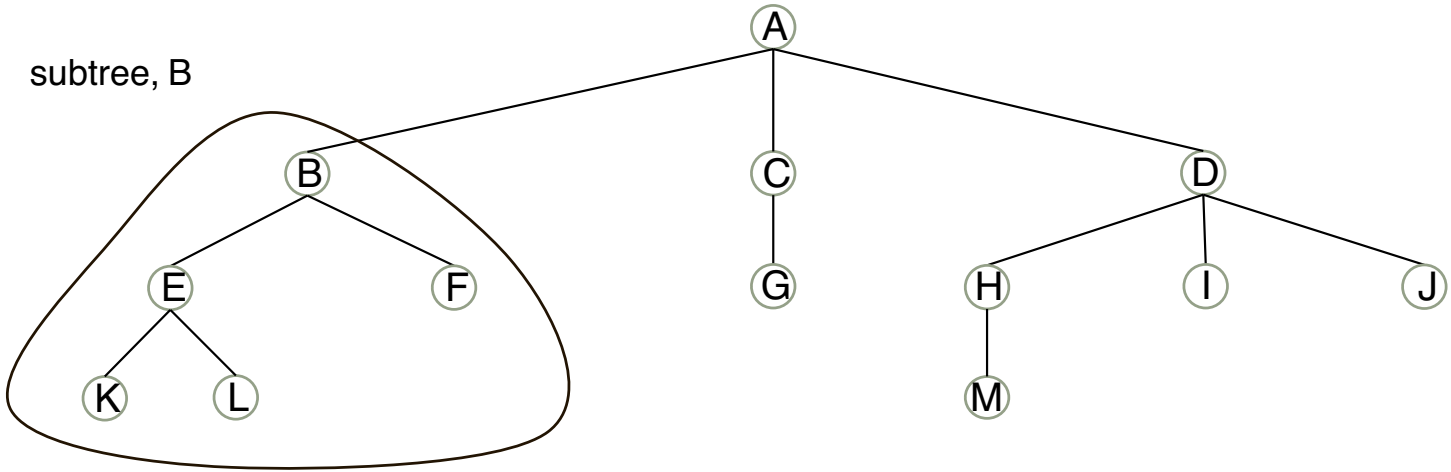
Left child-right sibling representation

Representation as a degree-two tree

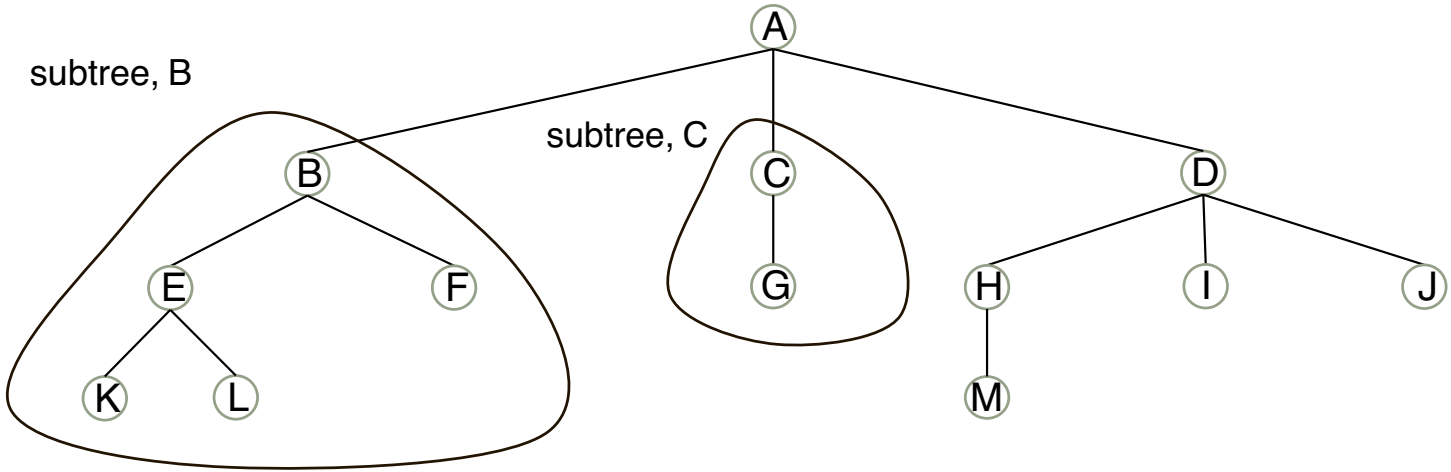
Tree: Example



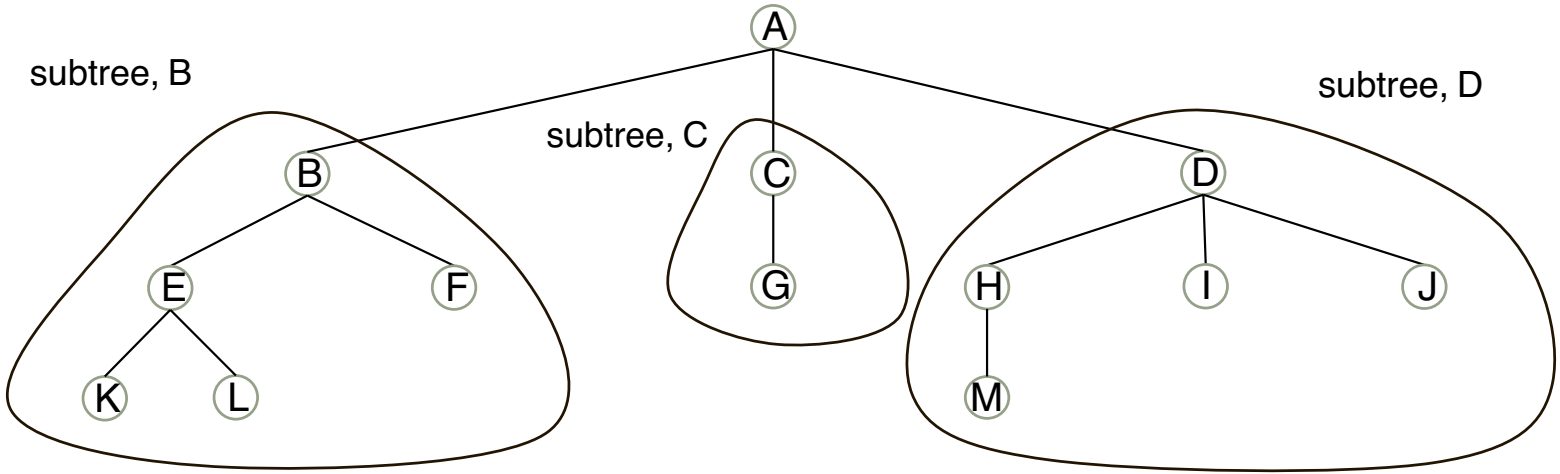
(A (B (E (K, L), F), C (G), D (H (M), I, J)))



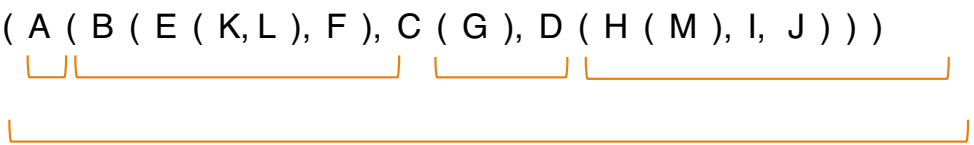
(A (B (E (K, L), F), C (G), D (H (M), I, J)))



(A (B (E (K, L), F), C (G), D (H (M), I, J)))



(A (B (E (K, L), F), C (G), D (H (M), I, J)))



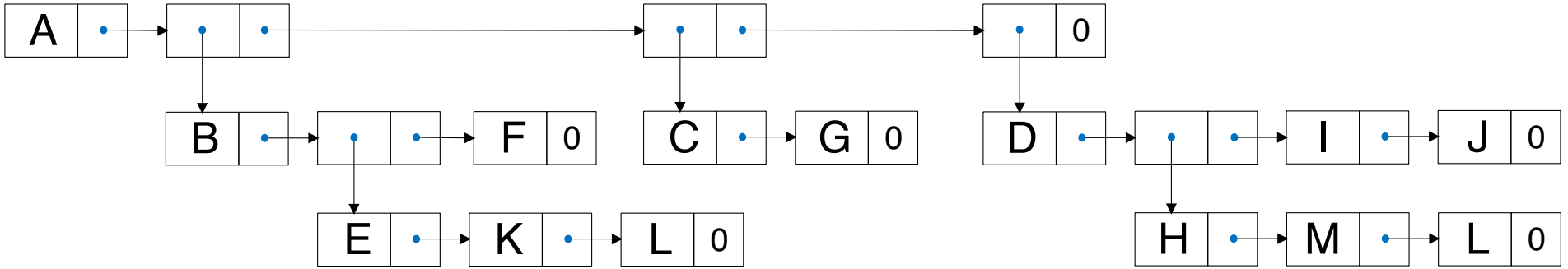
Root: A
Subtree root: B, C, D
String representation: (A (B, C, D)) ? (Root (Child node in sibling order separated by commas))

B ? (B (E, F)); E (K, L) ? (B (E (K, L), F))
C ? (C (G))
D ? (D (H, I, J)); H (M) ? (D (H (M), I, J))

A ? (A (B, C, D)) ? (A (B (E (K, L), F), C (G), D (H (M), I, J))

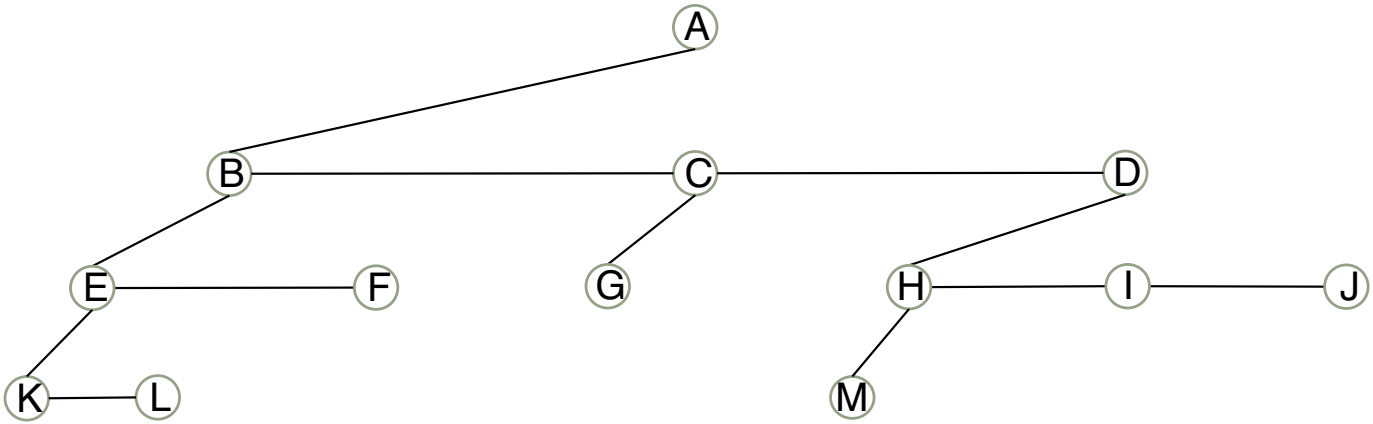
(A (B (E (K, L), F), C (G), D (H (M), I, J)))

List representation



(A (B (E (K, L), F), C (G), D (H (M), I, J)))

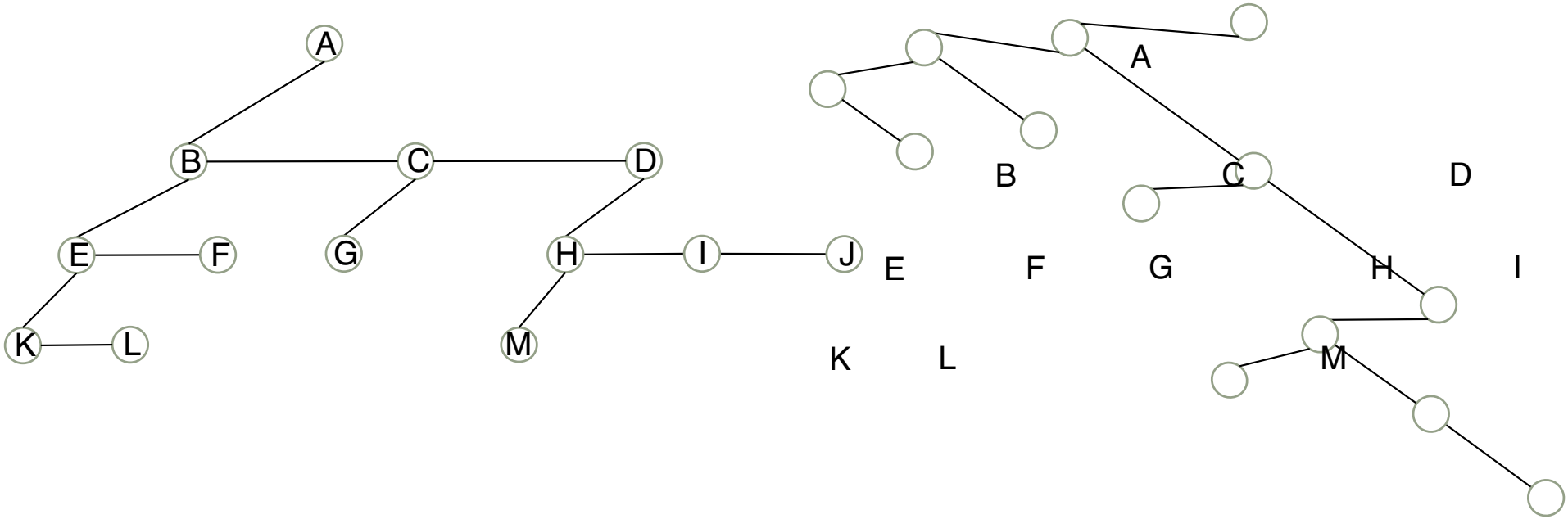
Left child-right sibling representation



(A (B (E (K, L), F), C (G), D (H (M), I, J)))

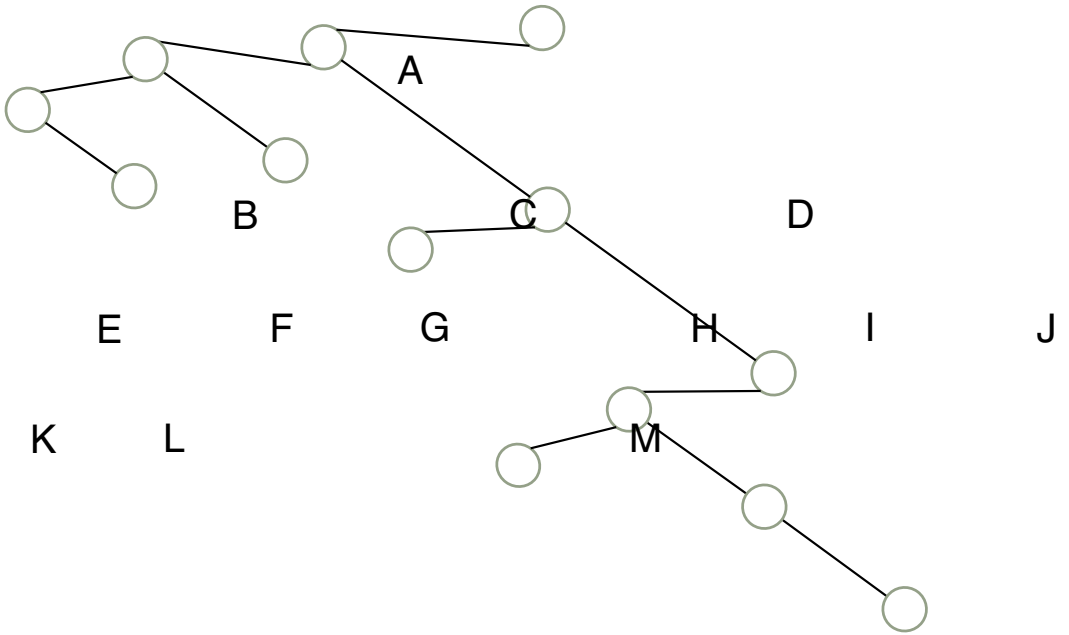
Left child-right sibling representation

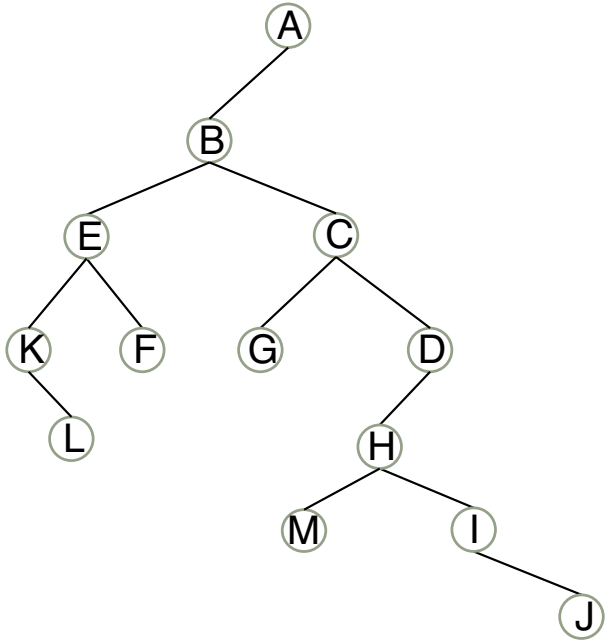
Rotate the tree



(A (B (E (K, L), F), C (G), D (H (M), I, J)))

Left child-right sibling representation → Representation as a Degree-Two Tree





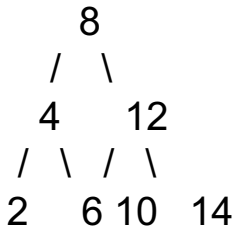
Issue: Balanced

A **balanced tree** ensures that the **height difference between subtrees** is minimized.

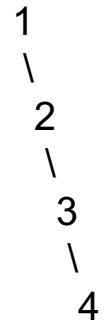
If the tree becomes too “deep” on one side, search and insertion operations degrade from **O(log n)** to **O(n)** — as bad as a linked list.

Balanced vs. Unbalanced (Shewed)

Balanced



Unbalanced (Skewed)



基於应用来决定

Balancing Strategies

Type	Balancing Rule	Application
AVL Tree	Height difference ≤ 1	Fast lookup
Red-Black Tree	Balanced via color constraints	C++ STL `map`, Linux kernel
B-Tree / B+Tree	Multi-level, multi-key nodes	Databases and file systems

Summary

Concept	Purpose	Complexity (Balanced Tree)
Balance	Maintain log-scale height for efficiency	O(log n) search/insertion
Traversal (DFT)	Recursive visiting order	O(n)
Traversal (BFT)	Level-order visiting	O(n)
Application	Decision Tree, BST, B-Tree, Red-Black, etc.	—

insert / delete

tree → adjust 不滿足 tree 的 status

→ 調整 (不滿足 ↑ 再調整)

(cost 高)

ADT: BinaryTree

ADT *BinaryTree* is

objects:

A finite set of nodes either empty or consisting of a root node, left *BinaryTree*, and right *BinaryTree*.

functions:

for all *bt, bt1, bt2* ∈ *BinaryTree*, *item* ∈ *element*

BinaryTree Create()	::=	creates an empty binary tree
Boolean IsEmpty(*bt*)	::=	**if** (*bt*==empty binary tree) **return** TRUE **else return** FALSE
BinaryTree MakeBT(*bt1*, *item*, *bt2*)	::=	return a binary tree whose left subtree is *bt1*, whose right subtree is *bt2*, and whose root node contains the data *item*
BinaryTree Lchild(*bt*)	::=	if (IsEmpty(*bt*)) return error else return the left subtree of *bt*
BinaryTree Rchild(*bt*)	::=	if (IsEmpty(*bt*)) return error else return the right subtree of *bt*
element Data(*bt*)	::=	if (IsEmpty(bt)) return error else return the data in the root node of *bt*

end *BinaryTree*

Binary Tree

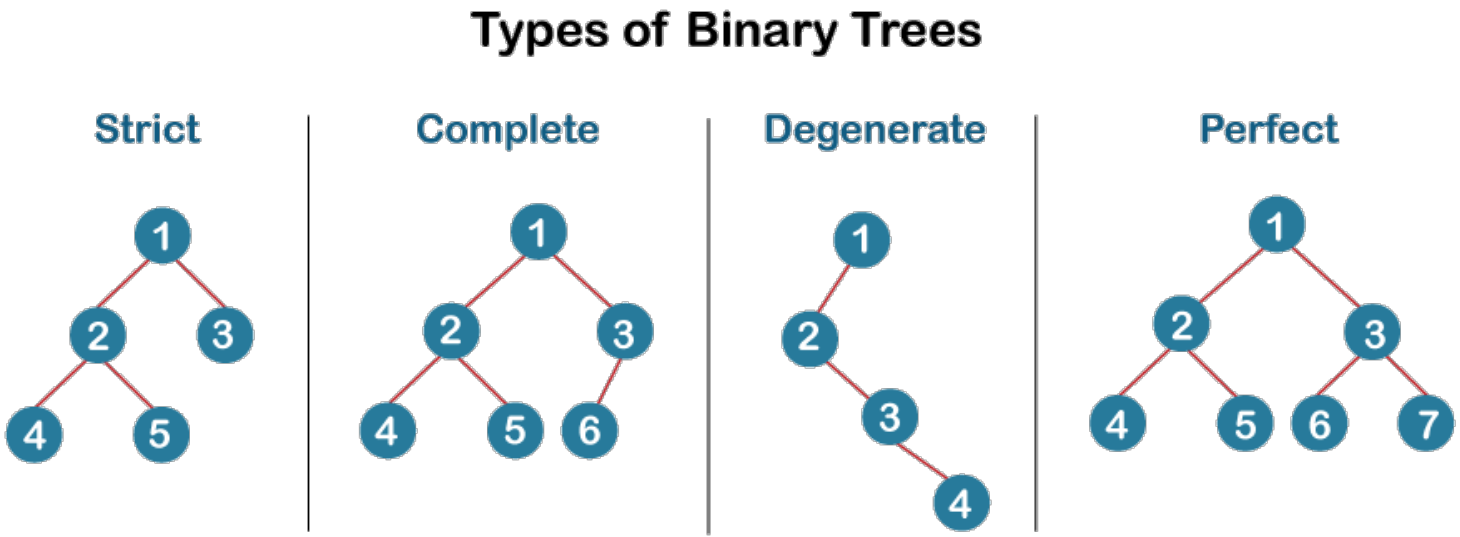


Image credit: https://www.linkedin.com/pulse/binary-trees-representation-traversals-implementation-riya-pandey-1kgff/

外觀上的定義

Binary Tree

A **binary tree** is a hierarchical data structure in which **each node has at most two children** .

Key Characteristics

- Each node can have:
  - 0 children (leaf)
  - 1 child
  - 2 children
- Children are conventionally named:
  - left child
  - right child
- There is no ordering constraint on the values in the tree.
- It is a shape-based definition, not value-based.

Binary Search Tree

A **Binary Search Tree** is a **binary tree with an ordering property** (left < parent < right).

BST Property

For every node:

- **All nodes in the left subtree** contain values **less than** the node's value.
- **All nodes in the right subtree** contain values **greater than** the node's value.
- This rule applies **recursively** to all subtrees.

Binary Search in Sorted Integer Array

Input integers: 52, 18, 82, 7, 69, 36, 95, 3, 11, 23, 27, 41, 60, 64, 78, 31, 45, 56, 73, 89

Input

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
52	18	82	7	69	36	95	3	11	23	27	41	60	64	78	31	45	56	73	89

Sorted

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
3	7	11	18	23	27	31	36	41	45	52	56	60	64	69	73	78	82	89	95

使用 array ⇒ 空間利用率↓

依續填值就好

random 後的字串

⇒ tree 會改變 (第一個按 route 改變)

Binary Tree

Input integers: 52, 18, 82, 7, 69, 36, 95, 3, 11, 23, 27, 41, 60, 64, 78, 31, 45, 56, 73, 89

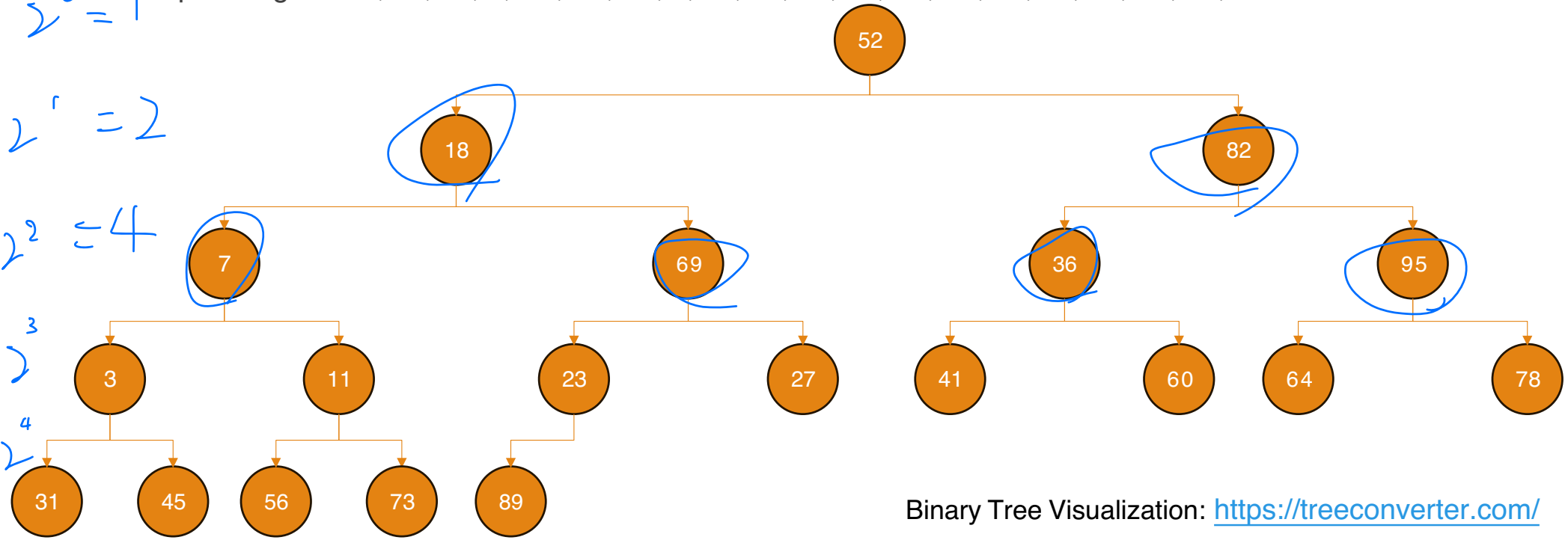
$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

2^3

2^4



Binary Tree Visualization: https://treeconverter.com/

Binary Tree in Array

Input integers: 52, 18, 82, 7, 69, 36, 95, 3, 11, 23, 27, 41, 60, 64, 78, 31, 45, 56, 73, 89

Input

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
52	18	82	7	69	36	95	3	11	23	27	41	60	64	78	31	45	56	73	89

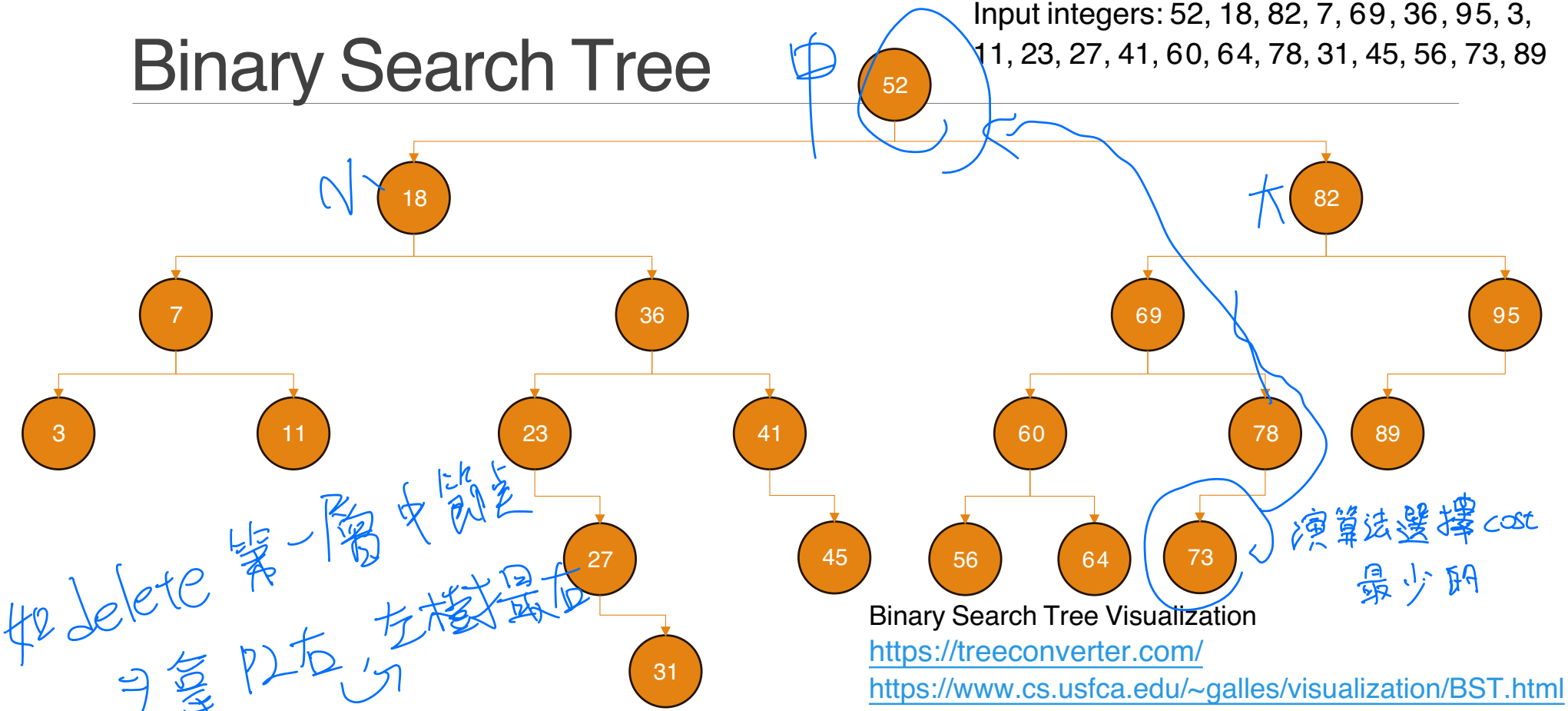
Binary Tree

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
52	18	82	7	69	36	95	3	11	23	27	41	60	64	78	31	45	56	73	89

小中大的組合，只跟父節點比，不是全部

Binary Search Tree

Input integers: 52, 18, 82, 7, 69, 36, 95, 3, 11, 23, 27, 41, 60, 64, 78, 31, 45, 56, 73, 89



Binary Search Tree Visualization
https://treeconverter.com/
https://www.cs.usfca.edu/~galles/visualization/BST.html

Binary Search Tree in Array

Input integers: 52, 18, 82, 7, 69, 36, 95, 3, 11, 23, 27, 41, 60, 64, 78, 31, 45, 56, 73, 89

Input

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
52	18	82	7	69	36	95	3	11	23	27	41	60	64	78	31	45	56	73	89

Binary Search Tree

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
52	18	82	7	36	69	95	3	11	23	41	60	78	89						

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
	27		45	56	64	73													

40	41	42	43
			31