Trees and Graphs

Davide Yi Xian Hu
Email: DavideYi.Hu@Polimi.it

POLITECNICO
MILANO 1863

Intro Presentation

Davide Yi Xian Hu

👉 Ph.D. Student Information Technology

👉 Mail: DavideYi.Hu@Polimi.it

👉 Research Interests:

- 🔎 Deep Learning Testing
- 🔎 Computer Vision Applications

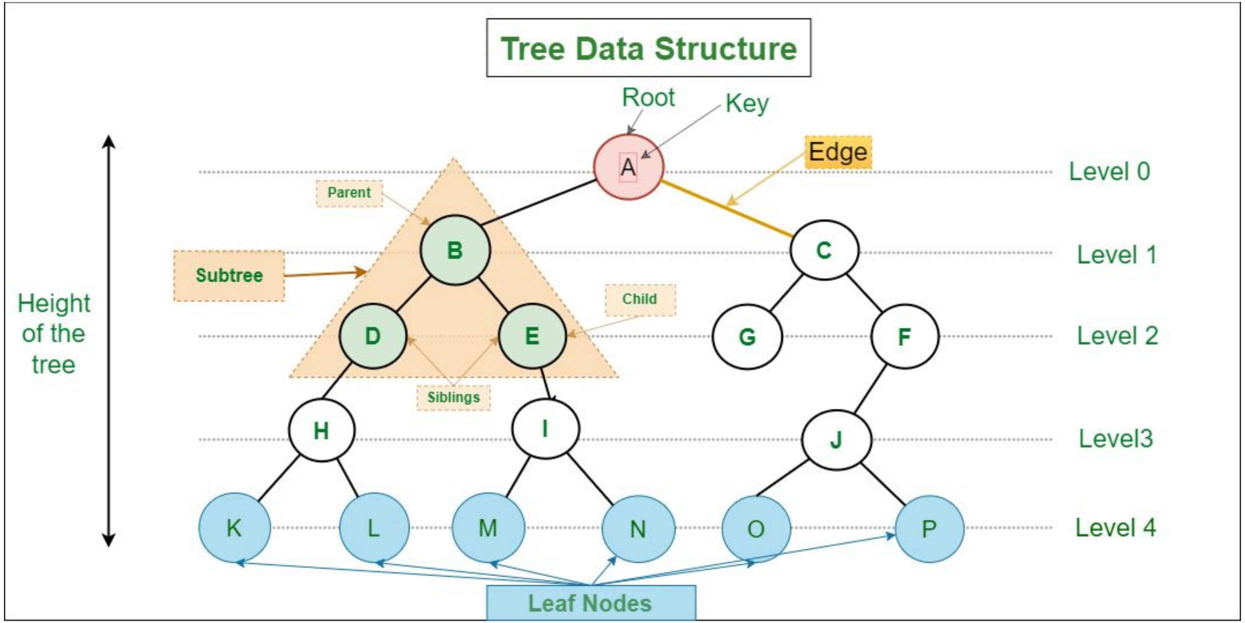
🌲 Trees

Tree Definition

Tree Data Structure

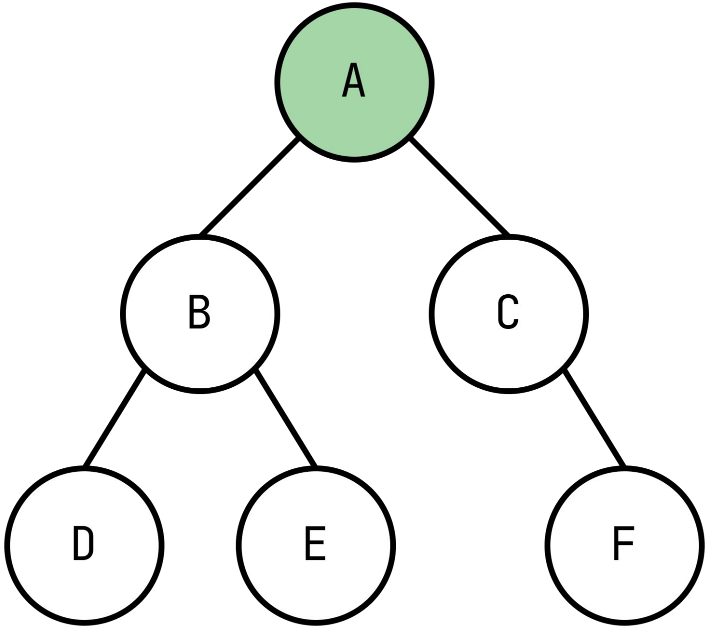
- 👉 A **Tree** 🎄 is a **Hierarchical** Data Structure with a Set of Nodes
- 👉 Each Node can have **many Children**
- 👉 Each Node have **exactly one Parent** (except for the root node)
- 👉 **No Cycles/Loops**

Example of Tree 🌳



BINARY TREE 🌴

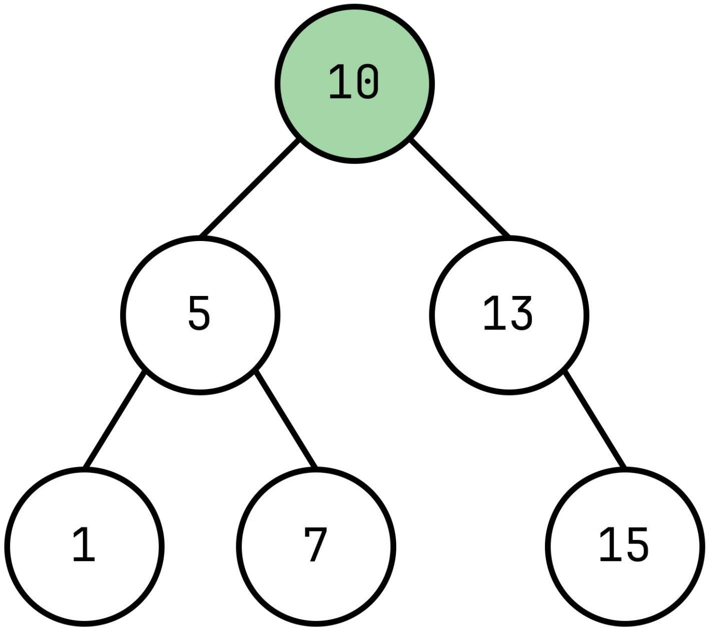
- 👉 ONE OF THE **MOST COMMONLY** USED TYPE OF TREES
- 👉 EACH NODE CAN HAVE AT MOST **TWO CHILDREN** (K-ARY TREE WITH K=2)



Binary Search Tree 🎋

The **Key** of a Node is:

- 👉 Greater than the Keys in the left subtree.
- 👉 Less than the Keys in the right subtree.

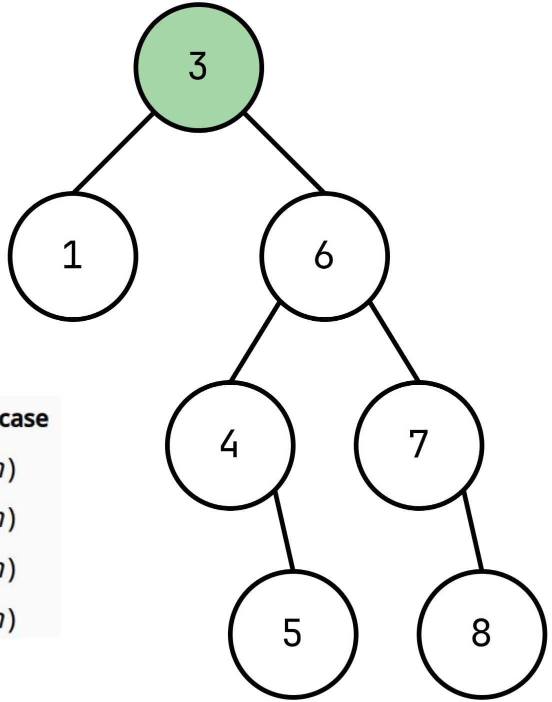


Balanced Trees 🌴

Binary Search Trees
Are **not Balanced**

Balanced Tree:
The **Height** of the
Left and the Right
subtree for each node
is either Zero or One

Algorithm	Average	Worst case
Space	O(*n*)	O(*n*)
Search	O(log *n*)	O(*n*)
Insert	O(log *n*)	O(*n*)
Delete	O(log *n*)	O(*n*)



Balanced Trees🌴

Binary Search Trees

Are **not Balanced**

Balanced Tree:

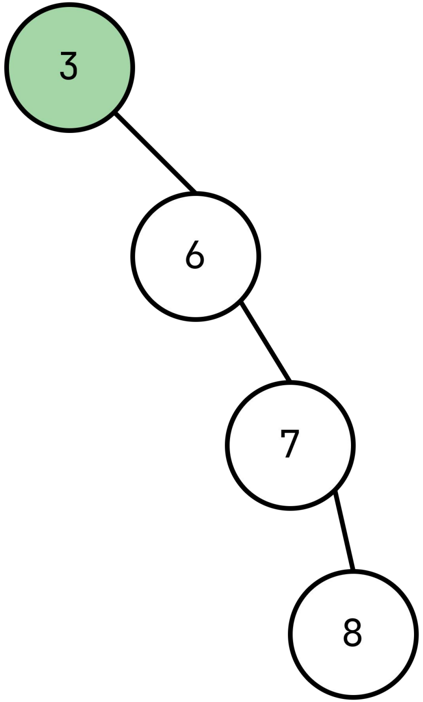
The **Height** of the

Left and the Right

subtree for each node

is either Zero or One

Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

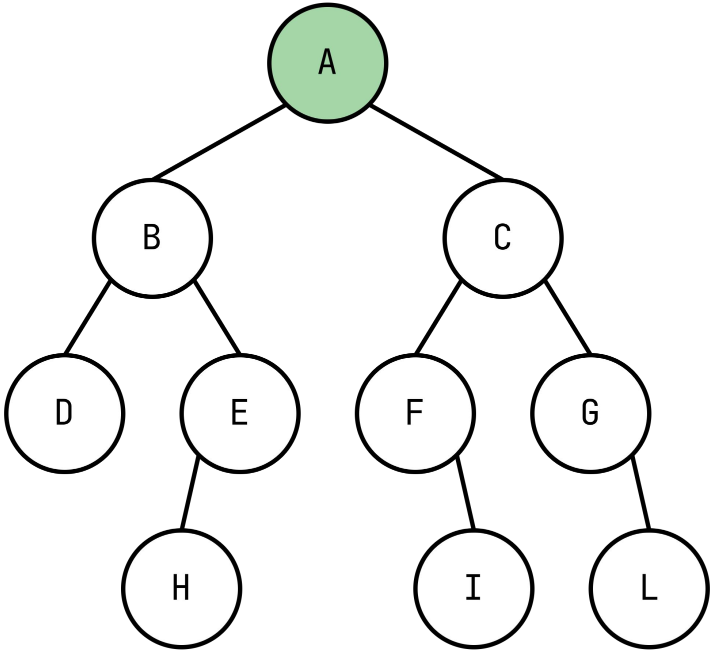


Balanced Search Trees 🌲

Examples of Balanced Trees:

- 👉 AVL (Binary) Tree
- 👉 Red-Black (Binary) Tree
- 👉 B-Tree (k-ary)

Search, Insertion and Deletion cost **O(log n)**



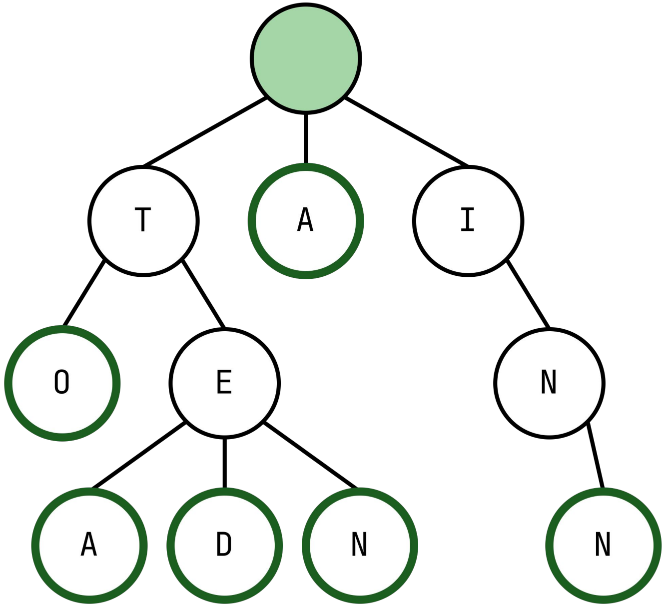
Trie 🌲

Known also as **Digital Tree** of **Prefix Tree**

String-indexed Lookup Data Structure

Applications:

- 👉 Autocomplete
- 👉 Full-text Search
- 👉 Blockchain (Merkle-Patricia)



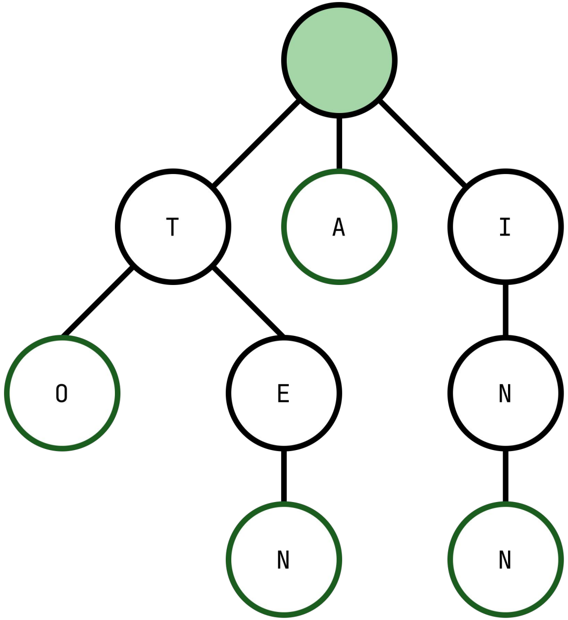
Trie 🌲

Known also as **Digital Tree** of **Prefix Tree**

String-indexed Lookup Data Structure

Applications:

- 👉 Autocomplete
- 👉 Full-text Search
- 👉 Blockchain (Merkle-Patricia)



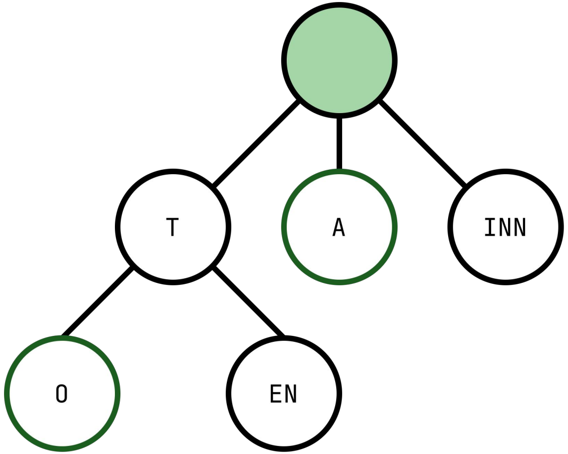
Trie 🌲

Known also as **Digital Tree** of **Prefix Tree**

String-indexed Lookup Data Structure

Applications:

- 👉 Autocomplete
- 👉 Full-text Search
- 👉 Blockchain (Merkle-Patricia)



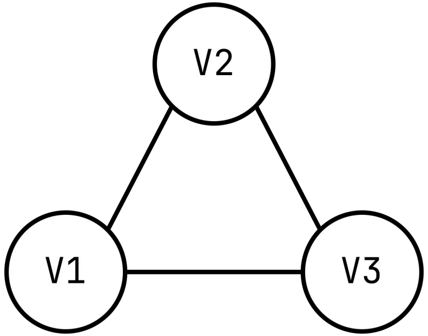
Compressed Trie

🍇 Graphs

Graphs Definition

Graph Data Structure

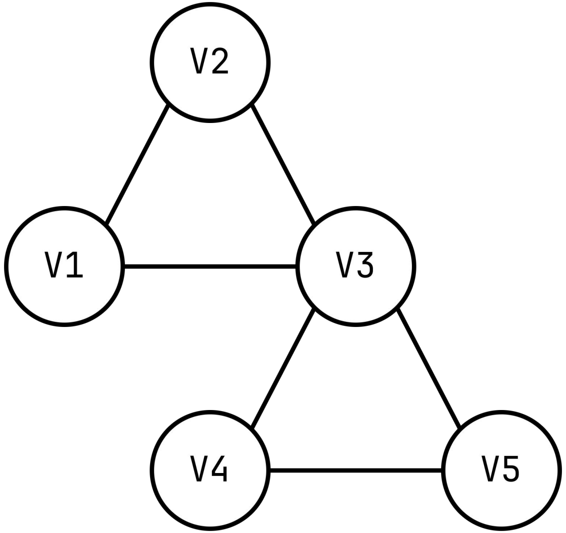
- A **Graph** 🍇 is a Data Structure with a Set of **Vertices** and **Edges**
- Graphs can be **Directed (DiGraph)** or **Undirected (Graph)**
- Both Vertices and Edges can hold Values
- Trees are a particular type of Graph



Graph Implementation

Graphs can be implemented as:

- 👉 **Adjacency List**
- 👉 **Adjacency Matrix**
- 👉 **Incidence Matrix**

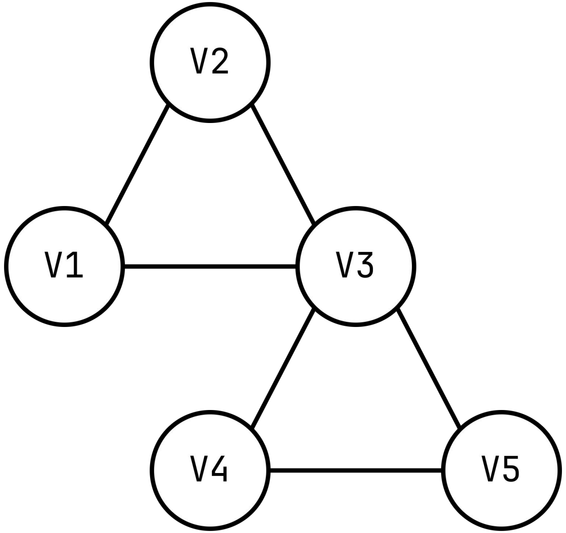


Graph Implementation

Adjacency List

List of Vertices and
Each Vertex has a list of
Adjacent Vertices

```
[ v1 [v2, v3],
  v2 [v1, v3],
  v3 [v1, v2, v3, v4],
  v4 [v3, v5]
  v5 [v3, v4]
]
```



Graph Implementation

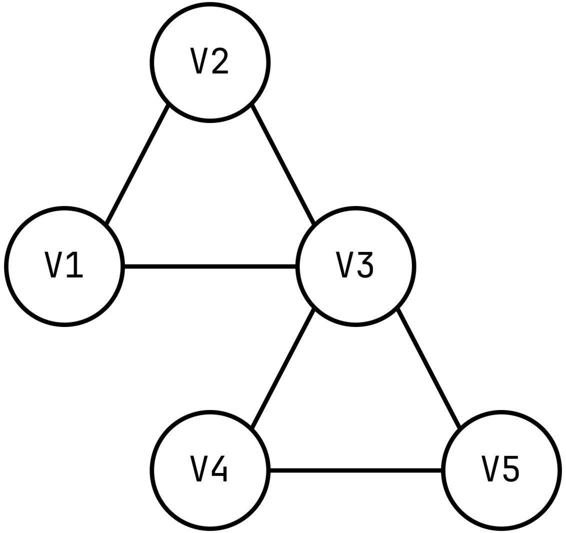
Adjacency Matrix

Two-dimensional Matrix

Rows represents **Source** vertices

Columns represents **Destination** Vertices

	V1	V2	V3	V4	V5
V1	-	X	X	0	0
V2	X	-	X	0	0
V3	X	X	-	X	X
V4	0	0	X	-	X
V5	0	0	X	X	-



Graph Implementation

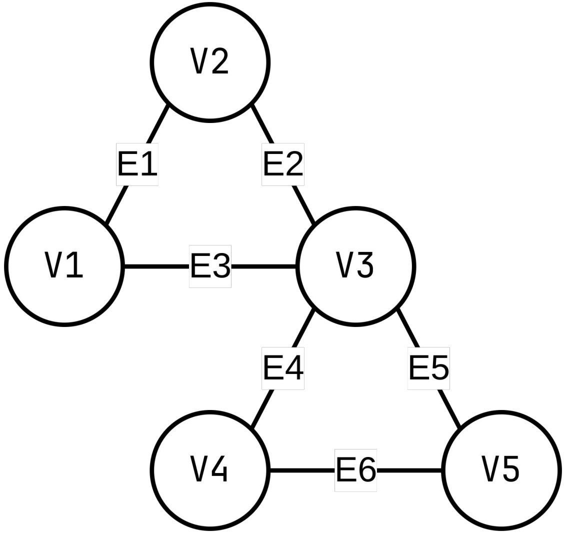
Incidence Matrix

Two-dimensional Matrix

Rows represents **Vertices**

Columns represents **Edges**

	E1	E2	E3	E4	E5	E6
V1	X	0	X	0	0	0
V2	X	X	0	0	0	0
V3	0	X	X	X	0	X
V4	0	0	0	X	X	0
V5	0	0	0	0	X	X



Graph Implementation

	Adjacency list	Adjacency matrix	Incidence matrix
Store graph	$O(\|V\|+\|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Add vertex	$O(1)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Add edge	$O(1)$	$O(1)$	$O(\|V\| \cdot \|E\|)$
Remove vertex	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Remove edge	$O(\|V\|)$	$O(1)$	$O(\|V\| \cdot \|E\|)$
Are vertices *x* and *y* adjacent (assuming that their storage positions are known)?	$O(\|V\|)$	$O(1)$	$O(\|E\|)$
Remarks	Slow to remove vertices and edges, because it needs to find all vertices or edges	Slow to add or remove vertices, because matrix must be resized/copied	Slow to add or remove vertices and edges, because matrix must be resized/copied

Graph Compressed Representation

Sparse Matrix

Is the most efficient way to represent a Matrix with lot of Zeros

Fast Construction:

LIL - **List of Lists (Adjacency List)**

DOK - **Dictionary of Keys**

COO - **Coordinate List**

(Fast Data Access):

CSR - **Compressed Sparse Row**

CSC - **Compressed Sparse Column**

GRAPH COMPRESSED REPRESENTATION

LIL

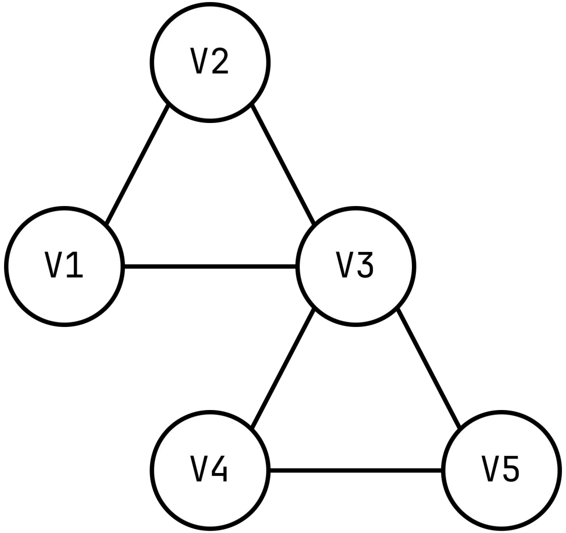
```
[ v1 [v2, v3],
  v2 [v1, v3],
  v3 [v1, v2, v3, v4],
  v4 [v3, v5]
  v5 [v3, v4]
]
```

DOK

```
{ (v1, v2),
  (v1, v3),
  (v2, v3),
  (v3, v4),
  (v3, v5),
  (v4, v5)}
```

COO

```
[ (v1, v2),
  (v1, v3),
  (v2, v3),
  (v3, v4),
  (v3, v5),
  (v4, v5)]
```



Search Algorithms

Search Algorithms

Search Algorithms

- 👉 They are used to Traverse or Explore a graph or tree in order to find a specific element or path.
- 👉 The most popular Algorithms are: **DFS (Depth First Search)** and **BFS (breadth First Search)**.

DFS

Depth First Search

Depth First Search

- 👉 Uses a stack data structure.
- 👉 Visits nodes depth-first (i.e., goes as deep as possible along each branch before **backtracking**).
- 👉 Good for traversing an entire tree or graph.
- 👉 Can get stuck in an infinite loop if the graph has cycles.

POLITECNICO MILANO 1863

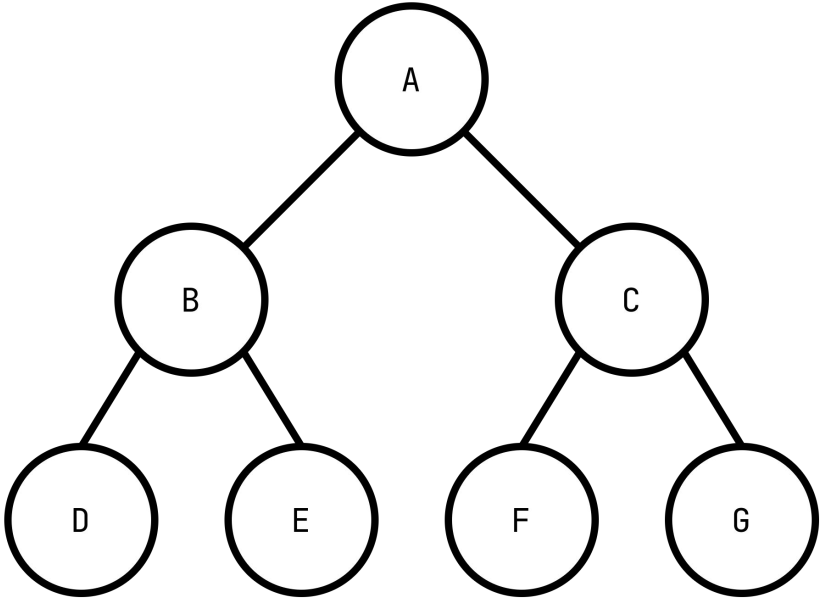
Depth First Search

Example:

Find a Path from A to F

Visited: [ ]

Stack: [ ]



Depth First Search

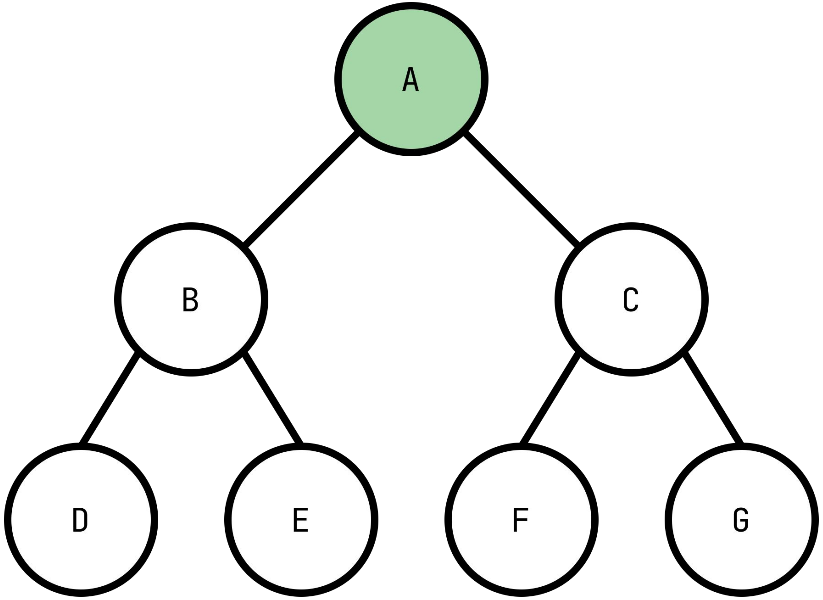
Example:

Find a path from A to F

Visit Node A

Visited: [A]

Stack: [B, C]



Depth First Search

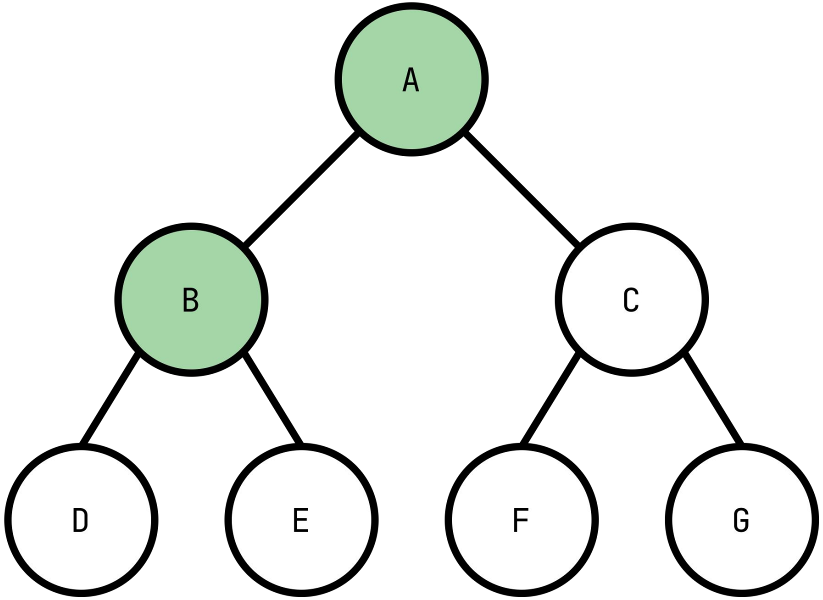
Example:

Find a Path from A to F

Visit Node B

Visited: [A, B]

Stack: [D, E, C]



Depth First Search

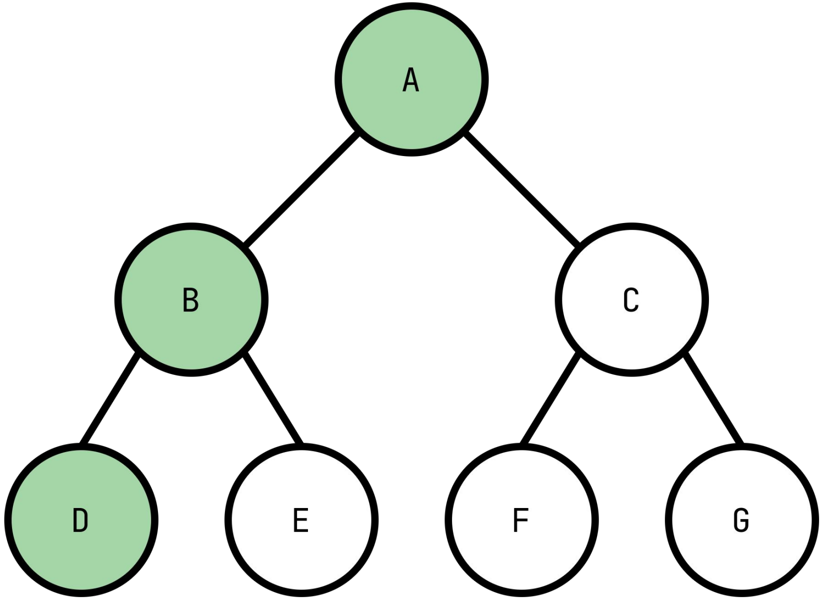
Example:

Find a Path from A to F

Visit Node D

Visited: [A, B, D]

Stack: [E, C]



Depth First Search

Example:

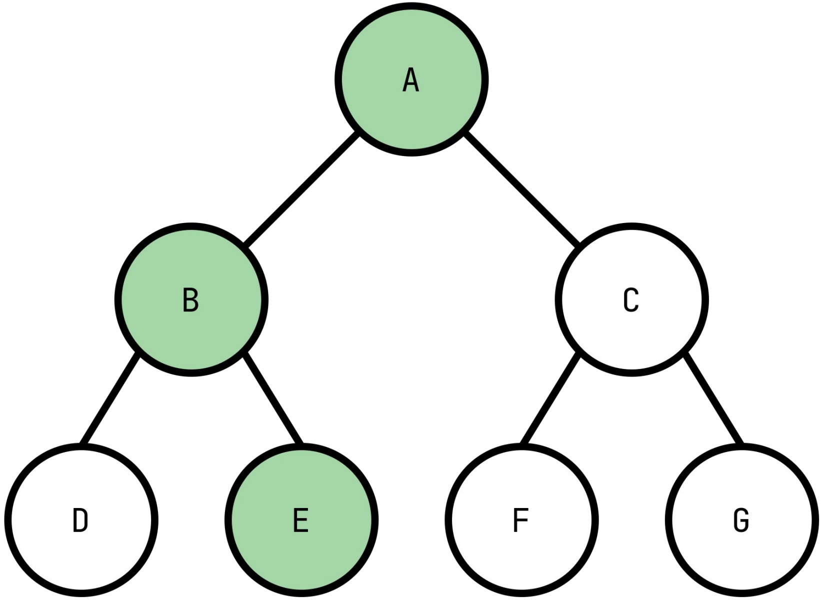
Find a Path from A to F

Backtracking

And Visit Node E

Visited: [A, B, D, E]

Stack: [C]



Depth First Search

Example:

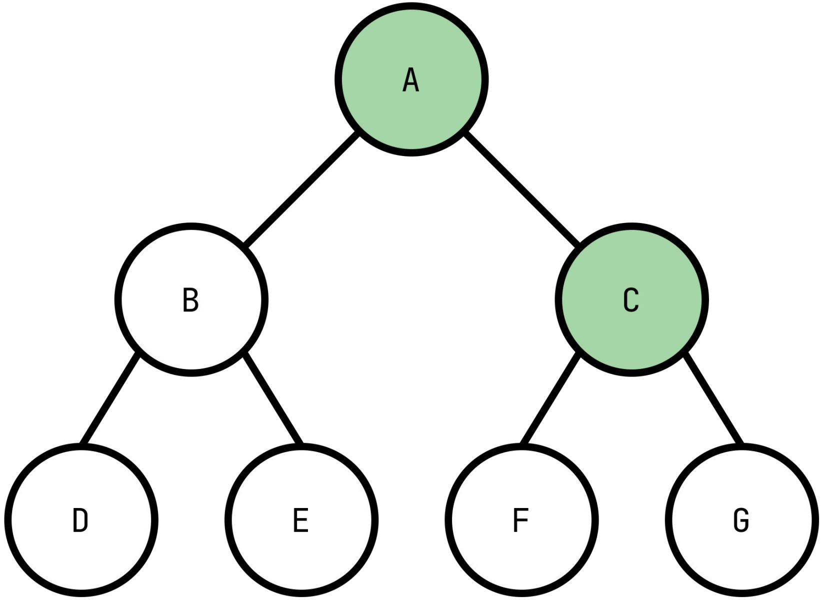
Find a Path from A to F

Backtracking

And Visit Node C

Visited: [A, B, D, E, C]

Stack: [F, G]



Depth First Search

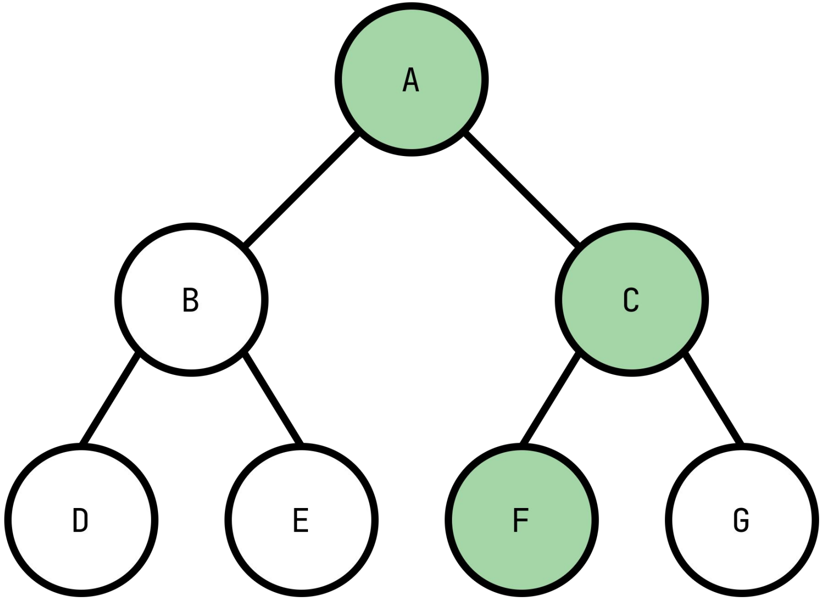
Example:

Find a path from A to F

Visit Node F

Visited: [A, B, D, E, C, F]

Stack: [G]



BFS
Breadth First Search

Breadth First Search

- 👉 Uses a queue data structure.
- 👉 Visits nodes breadth-first (i.e., explores all the neighbors of a node before moving on to the next level).
- 👉 Good for finding the shortest path between two nodes.
- 👉 May use more memory than DFS if the graph is large.
- 👉 May be slower than DFS for traversing the entire tree or graph.

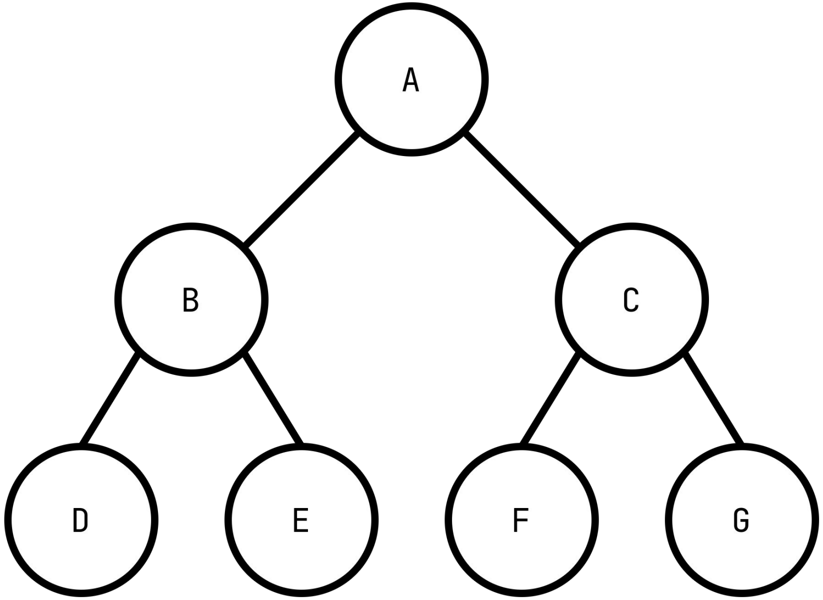
Breadth First Search

Example:

Find a Path from A to F

Visited: [ ]

Queue: [ ]



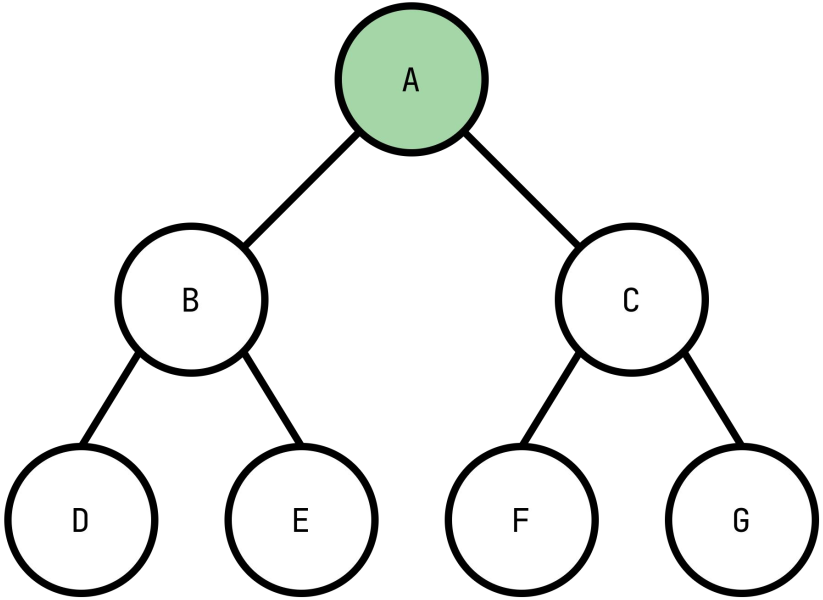
Breadth First Search

Example:

Find a Path from A to F

Visited: [A]

Queue: [B, C]



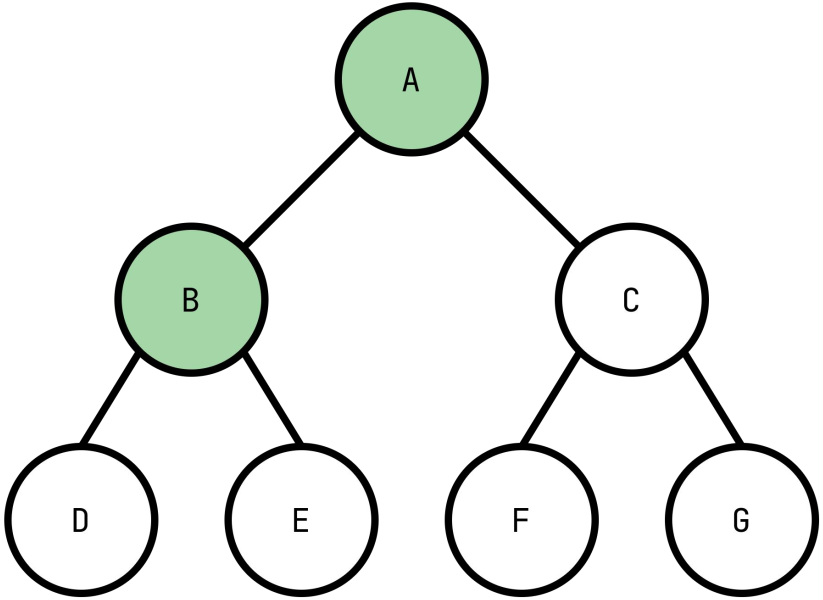
Breadth First Search

Example:

Find a Path from A to F

Visited: [A, B]

Queue: [C, D, E]



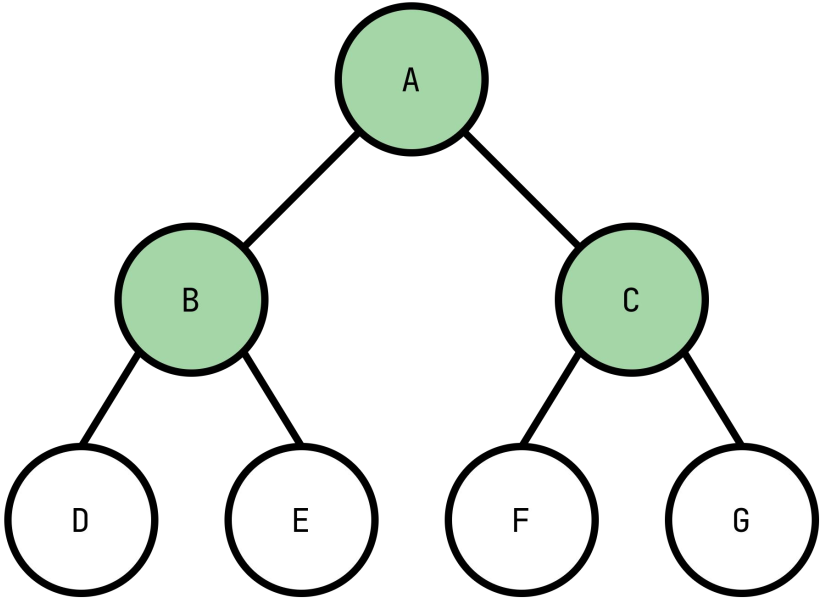
Breadth First Search

Example:

Find a Path from A to F

Visited: [A, B, C]

Queue: [D, E, F, G]



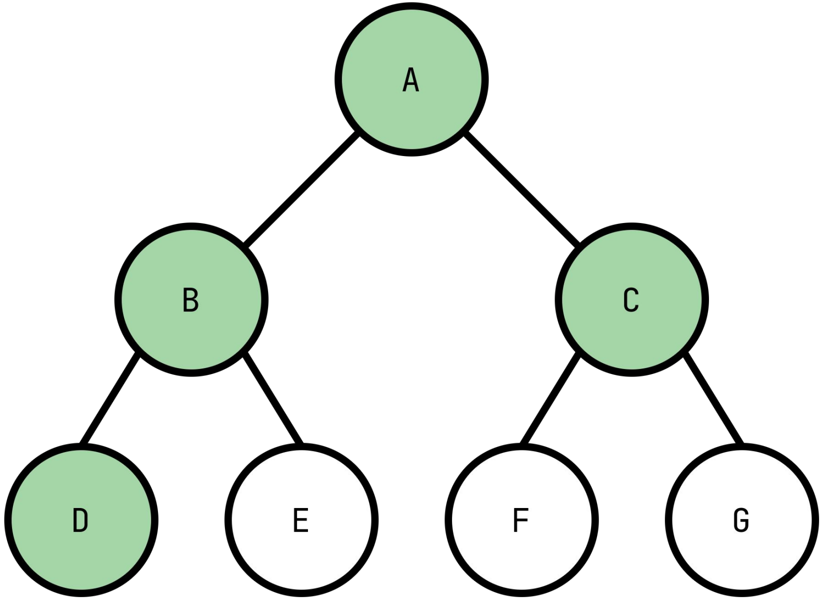
Breadth First Search

Example:

Find a Path from A to F

Visited: [A, B, C, D]

Queue: [E, F, G]



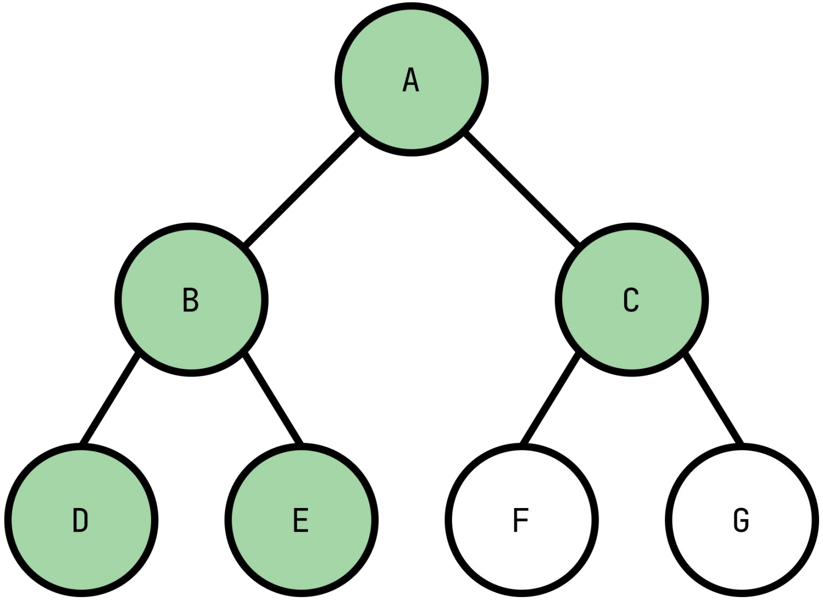
Breadth First Search

Example:

Find a Path from A to F

Visited: [A, B, C, D, E]

Queue: [F, G]



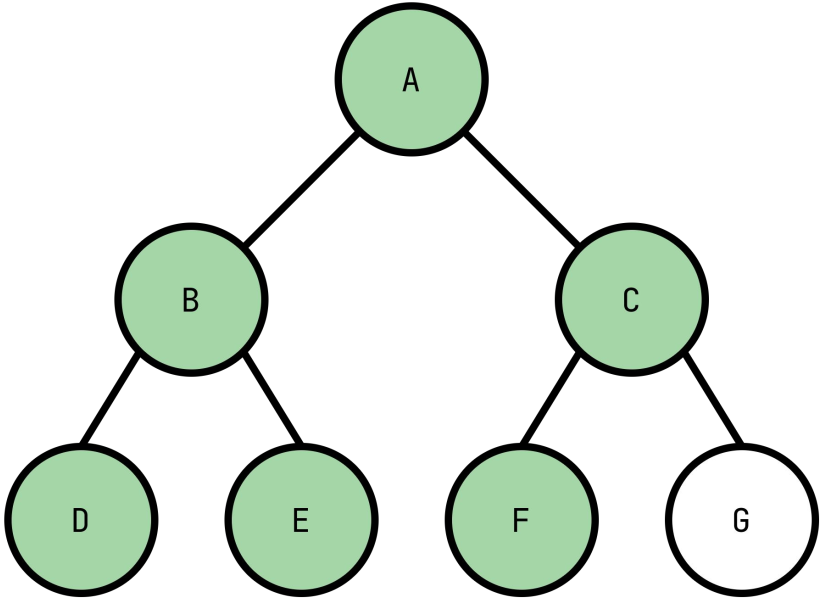
Breadth First Search

Example:

Find a path from A to F

Visited: [A, B, C, D, E, F]

Queue: [G]



Search Algorithms

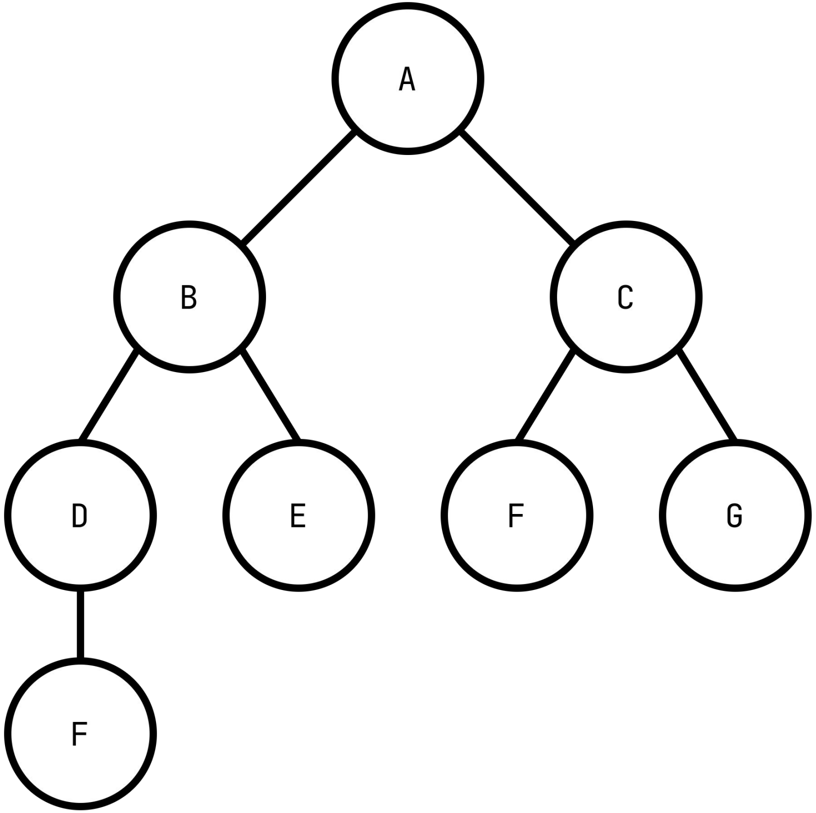
Complexity and Optimality

- Worst case complexities of both algorithms are the same.
  - **Time complexity**: $O(|V| + |E|)$
  - **Space complexity**: $O(|V|)$
- **BFS is optimal** if the path cost is a non-decreasing function of d(depth).

  Normally, BFS is applied when all the actions have the same cost.

Optimality Comparison

- 👉 Assume there are Two paths to Reach F
- 👉 Assume Each step Have the same Cost (for simplicity 1)



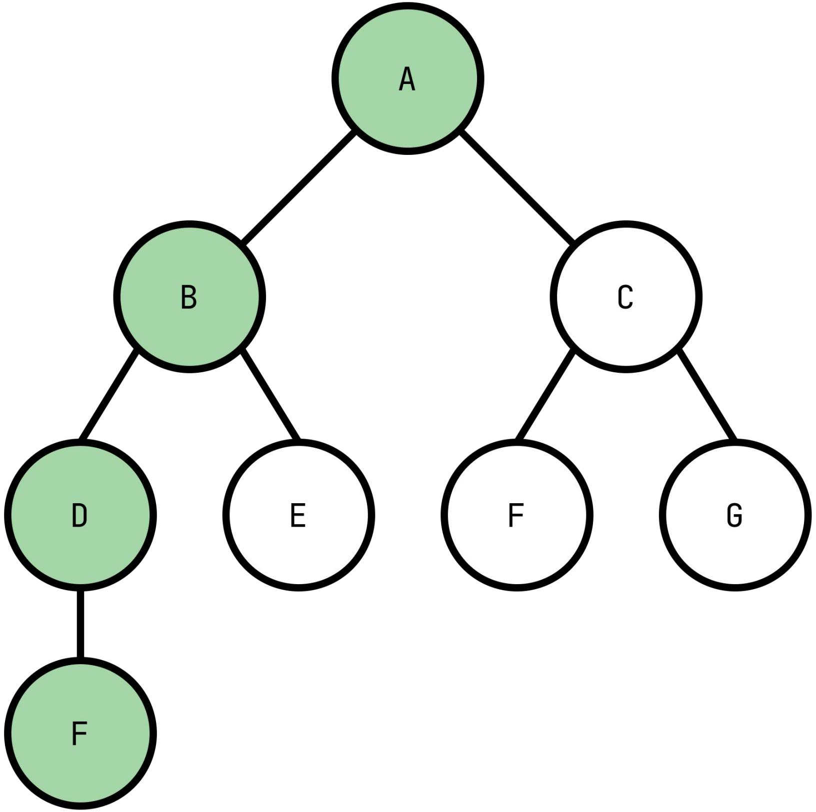
Optimality Comparison

Depth First Search

Path Found:

[A → B → D → F]

Path Cost: 4



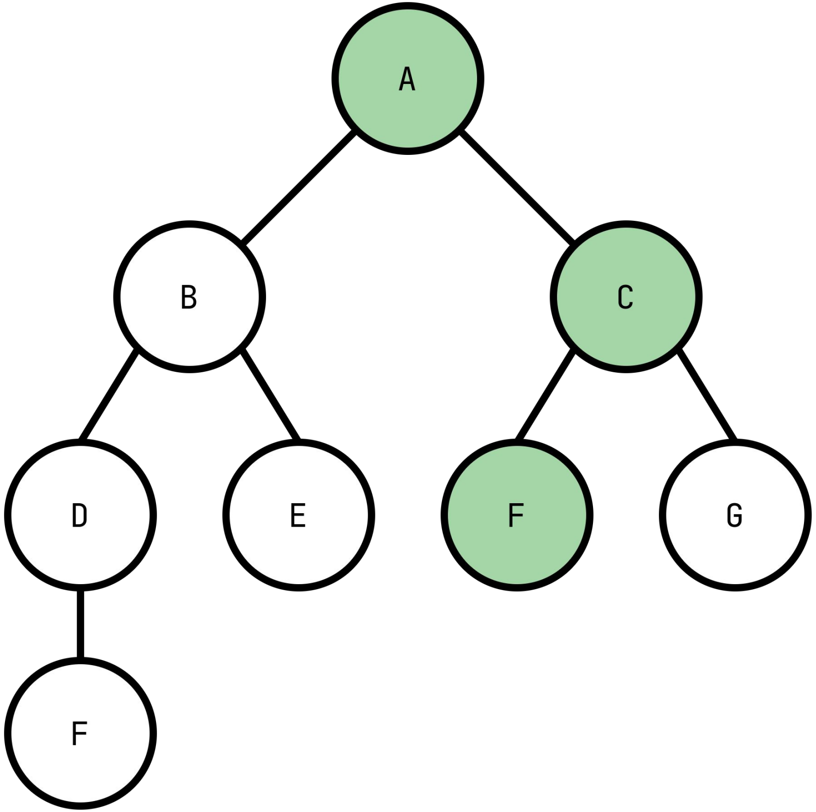
Optimality Comparison

Breadth First Search

Path Found:

[A → C → F]

Path Cost: 3



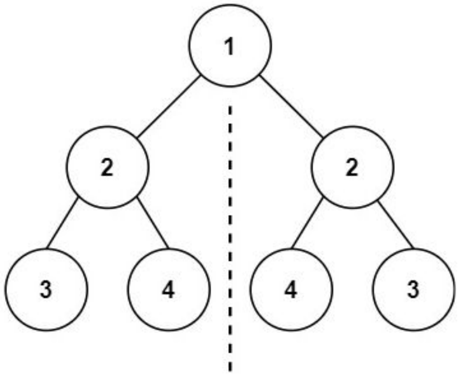
LeetCode

LeetCode Problem 1

101 Symmetric Tree

https://leetcode.com/problems/symmetric-tree/

Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).



POLITECNICO
MILANO 1863

LeetCode Problem 2

98 Validate Binary Tree

https://leetcode.com/problems/validate-binary-search-tree/

Given the root of a binary tree, determine if it is a valid binary search tree (BST). A valid BST is defined as follows:

- 👉 The left subtree of a node contains only nodes with keys less than the node's key.
- 👉 The right subtree of a node contains only nodes with keys greater than the node's key.
- 👉 Both the left and right subtrees must also be binary search trees.

LeetCode Problem 3

207 Course Schedule

https://leetcode.com/problems/validate-binary-search-tree/

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.

For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

Return true if you can finish all courses. Otherwise, return false.