자료구조

L06 Binary Tree

2022년 1학기

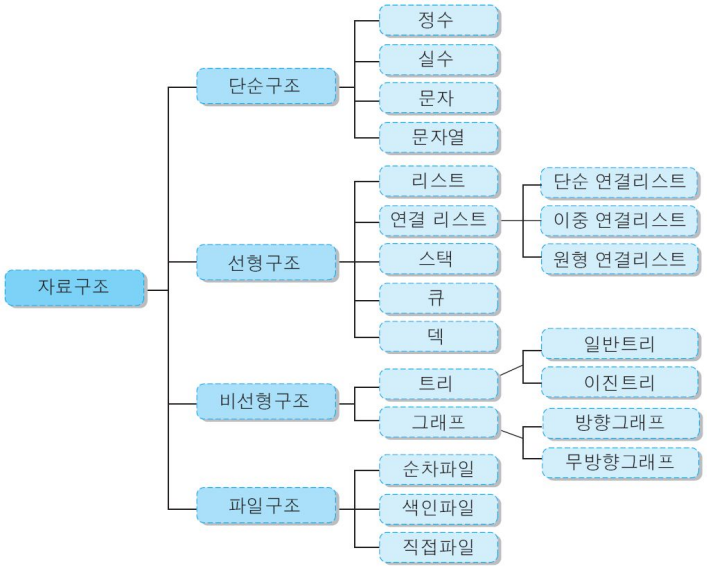
국민대학교 소프트웨어학부

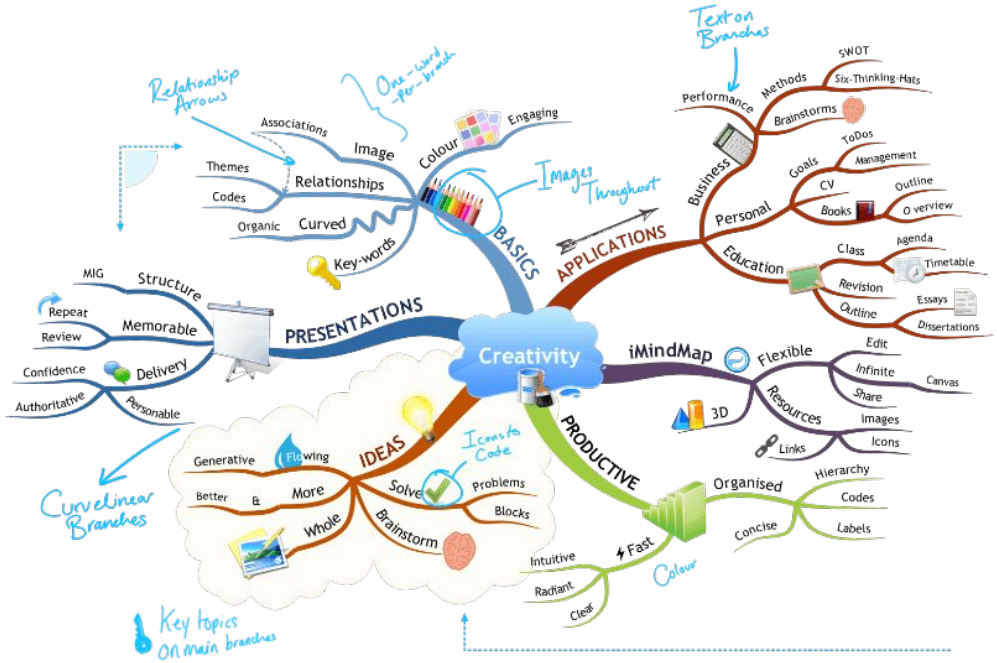
Outline

- **Trees and Terminology**
- Binary Trees
- Binary Tree Traversals

Tree Terminology

- **트리 구조**Tree Structure:
  자료의 계층적인 성질을 도식으로 표현하는 방법
  - 조직도, 토너먼트, 카테고리, 파일시스템, etc.

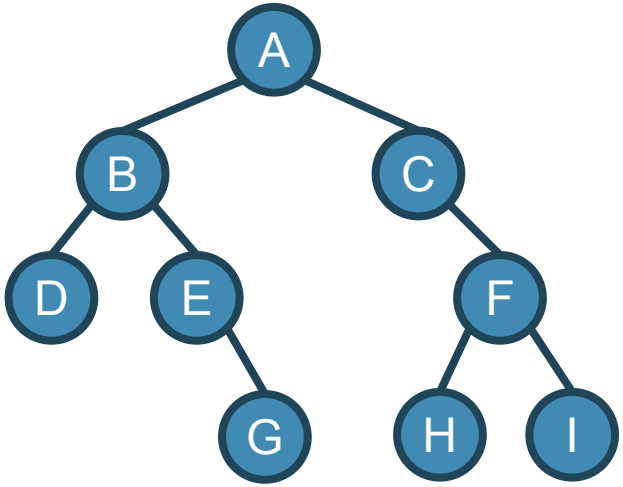


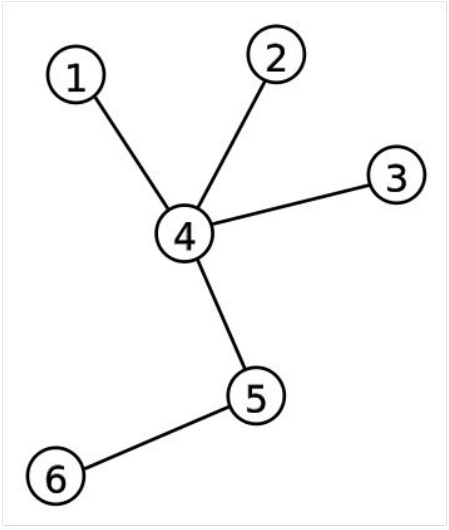


Tree Data Structure

- **루트가 있는 트리**Rooted Tree: 트리 구조를 **노드**와 **간선**의 **집합**으로 표현한 것

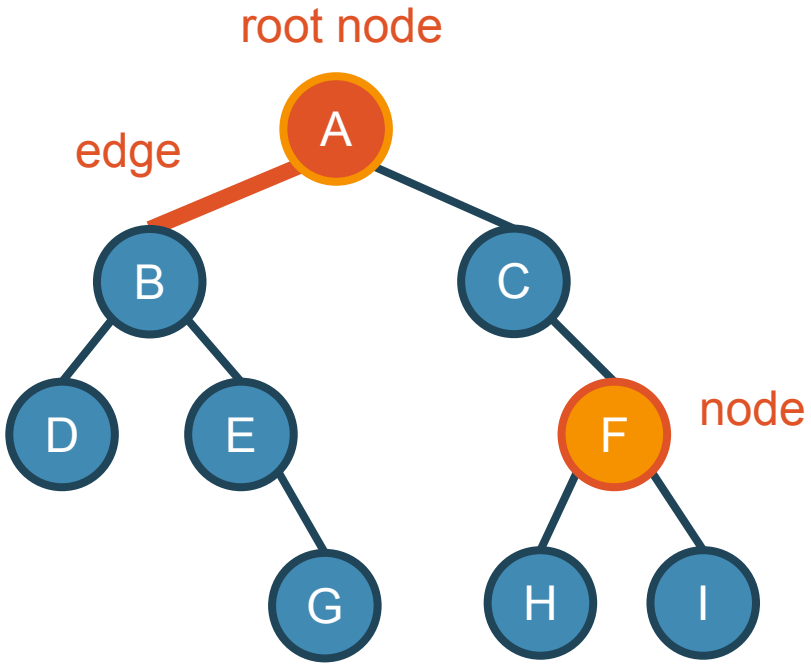
- **(일반적인) 트리**Tree: 루트가 없는 트리
  - Acyclic Undirected Connected Graph





Tree Terminology

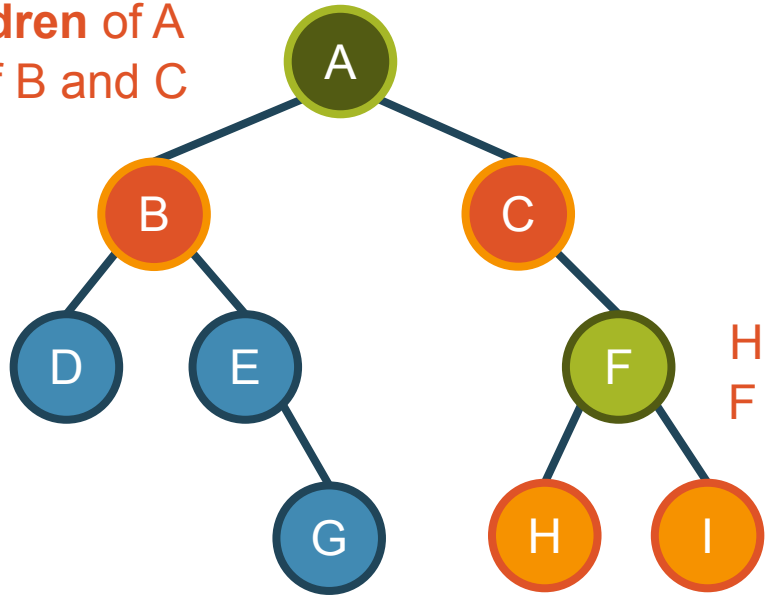
- **노드**Node: a unit
- **간선**Edge: 두 노드 사이의 연결
- **루트**Root: 가장 상위에 있는 노드



Tree Terminology

- **부모 노드**Parent: 노드 X의 부모 노드는 X와 간선으로 **연결된 노드 중 루트에 더 가까운 노드**
- **자식 노드**Child: 노드 X의 자식 노드는 X와 간선으로 **연결된 노드 중 루트에서 더 멀리 있는 노드**

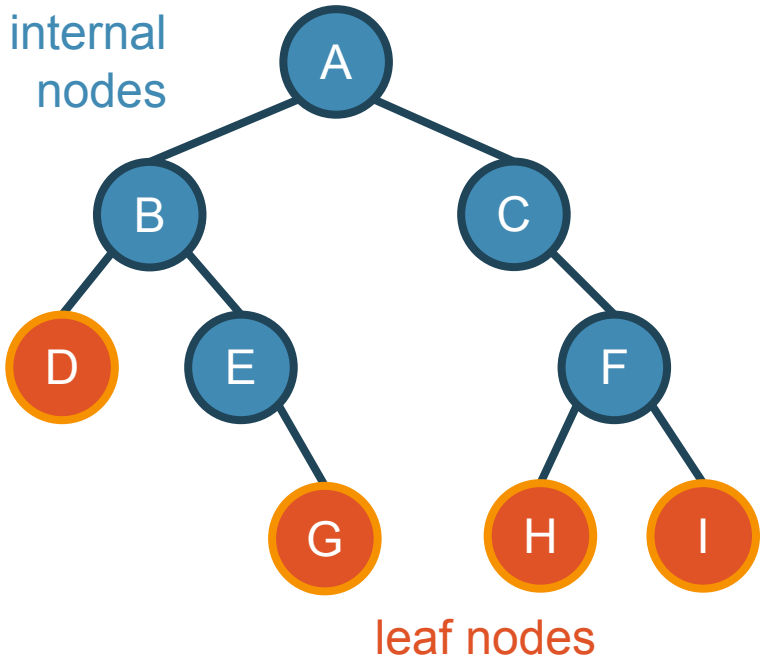
B and C are **children** of A
A is the **parent** of B and C



H and I are **children** of F
F is the **parent** of H and I

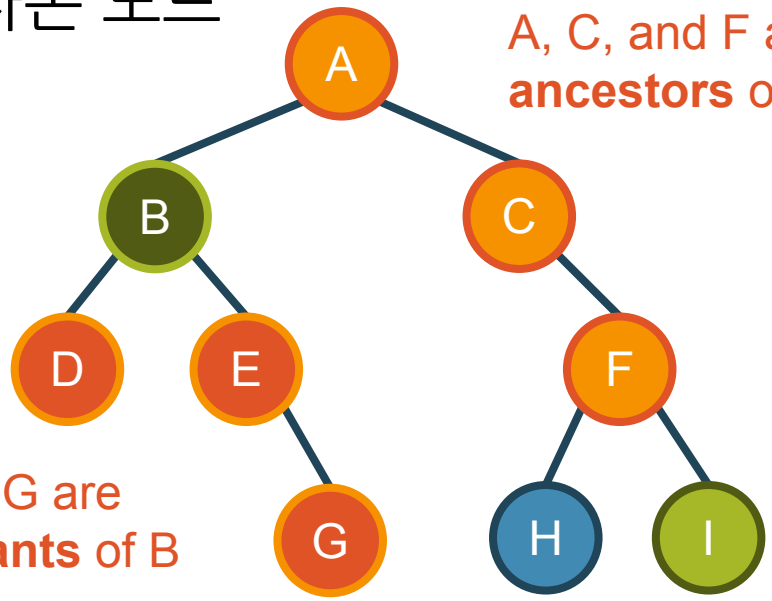
Tree Terminology

- 단말 노드Leaf Node: 자식이 없는 노드
- 내부 노드Internal node: 단말 노드가 아닌 노드



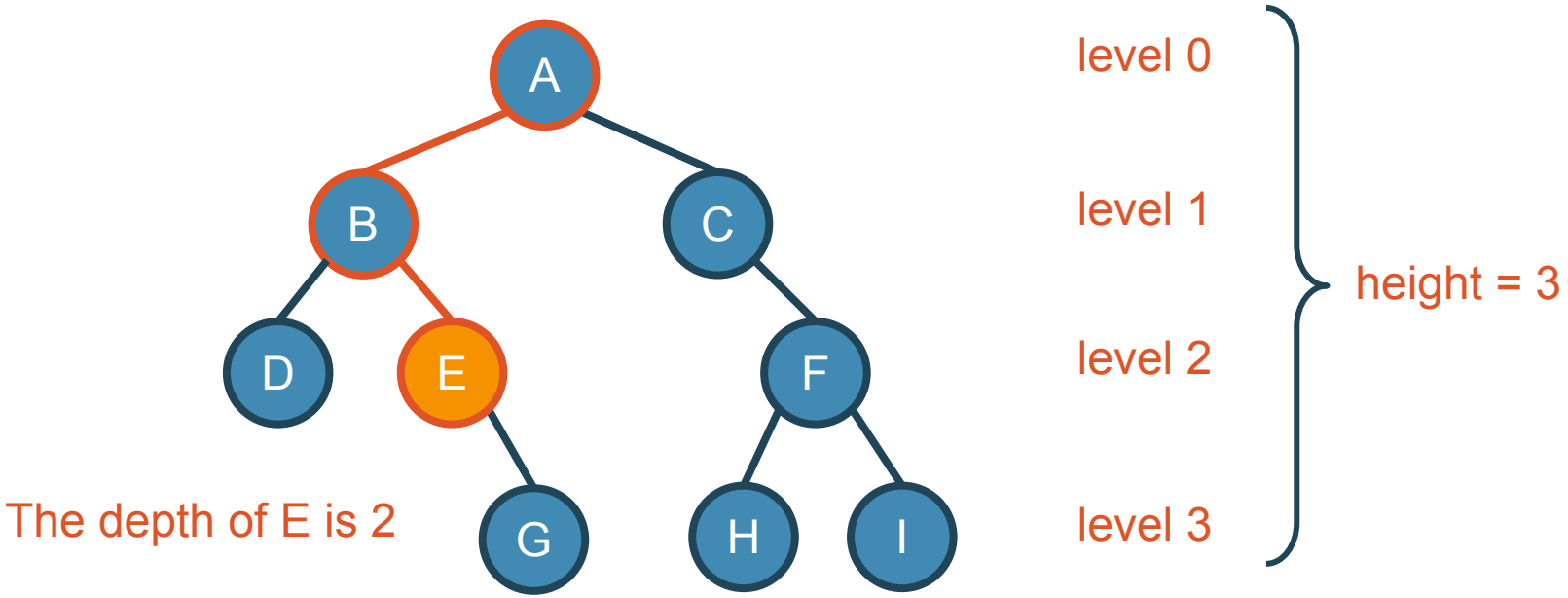
Tree Terminology

- **조상 노드**Ancestor:
  - 부모 노드는 조상 노드
  - 조상 노드의 부모 노드 역시 조상 노드
- **자손 노드**Descendant:
  - 자식 노드는 자손 노드
  - 자손 노드의 자식 노드 역시 자손 노드



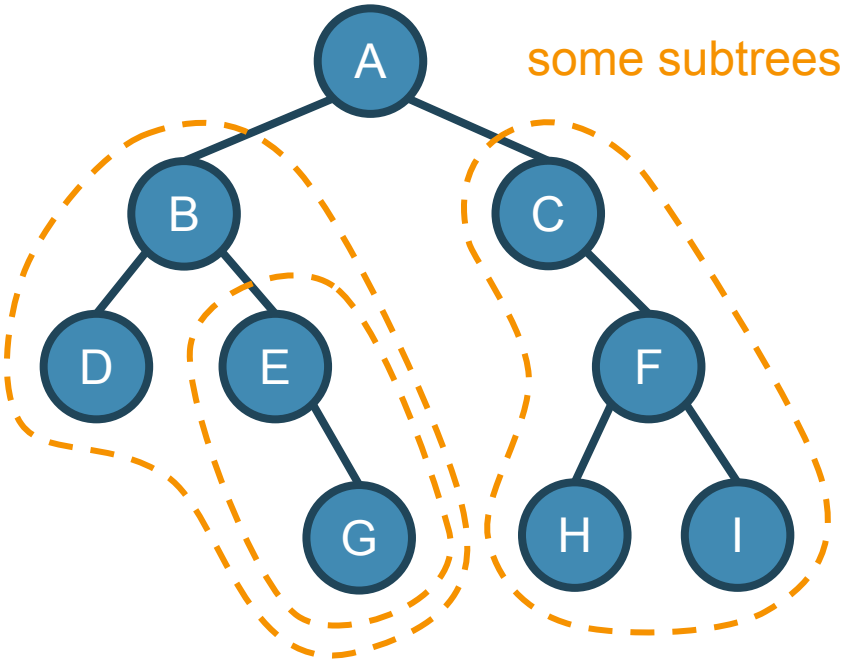
Tree Terminology

- **노드의 깊이**Depth of a node: 루트 노드와의 거리 (the depth of the root is 0)
- **레벨**Level: 깊이가 같은 노드들의 집합
- **트리의 높이**Height: 노드의 최대 깊이



Tree Terminology

- **서브트리**Subtree:
  한 노드와 그 노드의 모든 자손 노드를 포함하는 트리

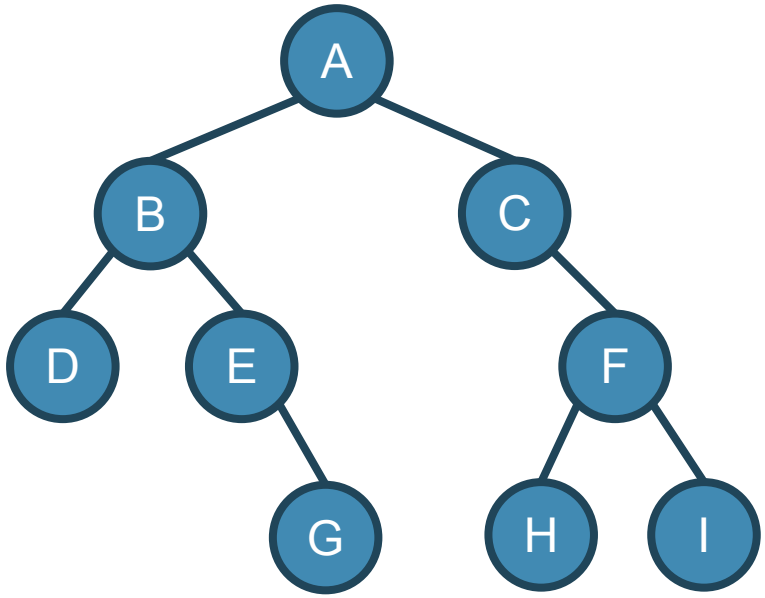


Outline

- Trees and Terminology
- **Binary Trees**
- Binary Tree Traversals

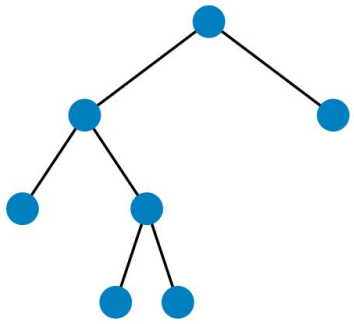
이진 트리 Binary Trees

- **이진 트리**: 각 노드가 자식 노드를 최대 두 개 까지만 가지는 트리. 두 자식 노드는 각 각 왼쪽 자식, 오른쪽 자식 이라고 부름.

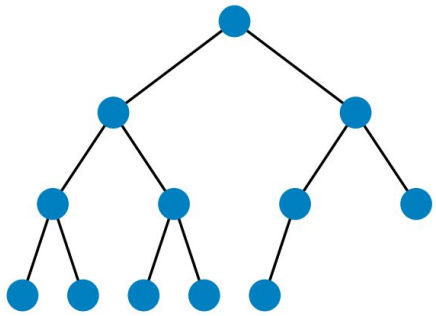


Full and Complete Binary Trees

- **정 이진 트리**Full binary tree: 각 노드의 자식 노드 수가 2 또는 0인 트리
- **완전 이진 트리**Complete binary tree:
  - 가장 깊은 레벨을 제외한 모든 레벨이 가득 차 있음
  - 마지막 레벨의 노드들은 가능한 왼쪽에 존재



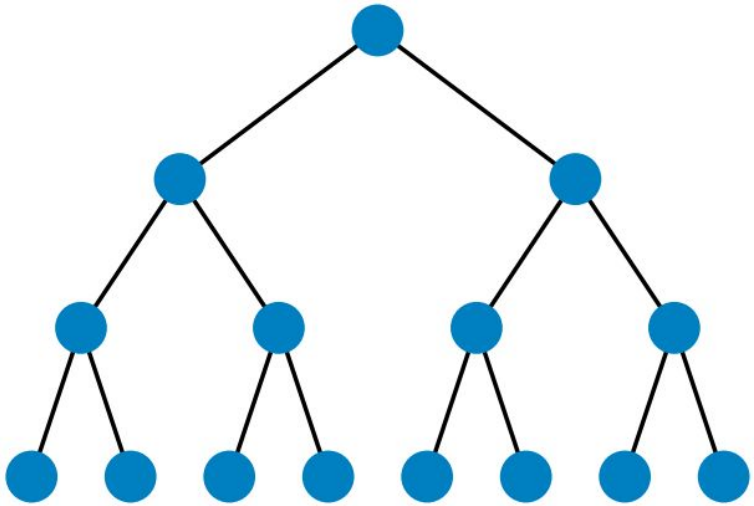
A full binary tree



A complete binary

Perfect Binary Tree

- **포화 이진 트리**Perfect binary tree:
  - 모든 단말 노드의 레벨이 같음
  - 모든 내부 노드의 자식의 수가 2임

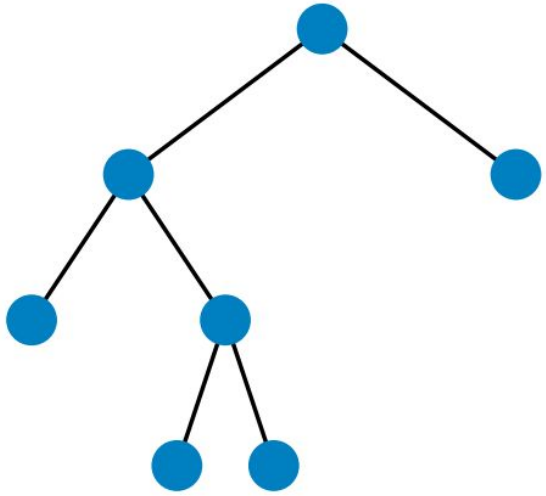


Full Binary Tree Theorem

정 이진 트리 정리

비어있지 않은 정 이진 트리의 리프 노드 수는 내부 노드의 수보다 한 개 많다.

- **증명** (by Induction):
  - **Base case**: 비어있지 않으면서 내부 노드가 없는 트리는 하나의 리프 노드 (루트 노드)를 갖는다.
  - **Induction Hypothesis**: n-1 개의 내부 노드를 갖는 모든 정 이진 트리는 n개의 리프 노드를 갖는다고 가정.



Full Binary Tree Theorem

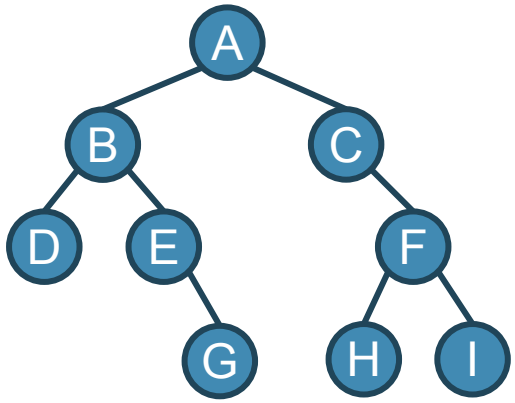
- **증명** (이어서 계속):
  - **Induction Step**:
    - n개의 내부 노드를 갖는 비어있지 않은 정 이진 트리 T가 주어졌을 때, 두 리프 노드를 자식 노드로 갖는 내부 노드 하나를 고르고 이 노드를 X라 하자. X의 두 자식 노드를 지운 트리를 T' 라 하자. 정의에 의해 T'는 정 이진 트리이다.
    - Induction Hypothesis에 의해 T'는 n개의 리프 노드를 갖는다.
    - 다시 X의 두 자식 노드를 복원한다. 내부 노드 개수는 n개 이며, 리프노드의 수는 n+1이 된다.

Full Binary Tree Corollary

정 이진 트리 정리의 따름 정리

비어있지 않은 이진 트리에서 null pointer의 수는 노드 수보다 1개 많다.

- **증명** (by Induction):
  - null pointer의 위치에 비어있는 노드를 넣어 리프 노드로 만들면, 정 이진 트리가 된다.



null pointers = 10
nodes = 9

Binary Tree Node ADT

```
/** ADT for binary tree nodes */
public interface BinNode<E> {
    /** Return and set the element value */
    public E element();
    public E setElement(E v);

    /** Return the left child */
    public BinNode<E> left();

    /** Return the right child */
    public BinNode<E> right();

    /** Return true if this is a leaf node */
    public boolean isLeaf();
}
```

Outline

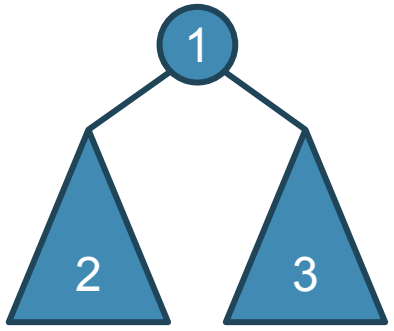
- Trees and Terminology
- Binary Trees
- **Binary Tree Traversals**

Traversals

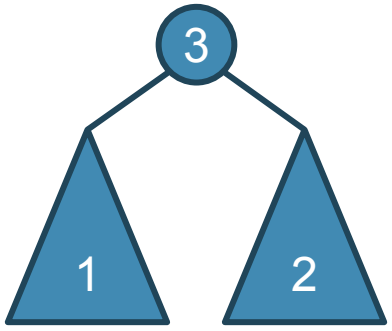
- **순회**Traversals: 순서대로 노드를 방문하는 과정
- **열거**Enumeration: 트리의 각 노드를 정확히 한번씩 나열하는 순회 방법

Binary Tree Traversals

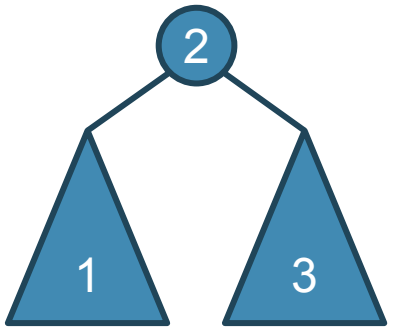
- **전위순회**Preorder traversal: 노드를 방문하고, 자식들을 순회한다.
- **후위순회**Postorder traversal: 자식들을 방문하고, 노드를 방문한다.
- **중위순회**Inorder traversal: 왼쪽 subtree를 방문하고, 노드를 방문하고, 오른쪽 subtree를 방문한다.



Preorder traversal



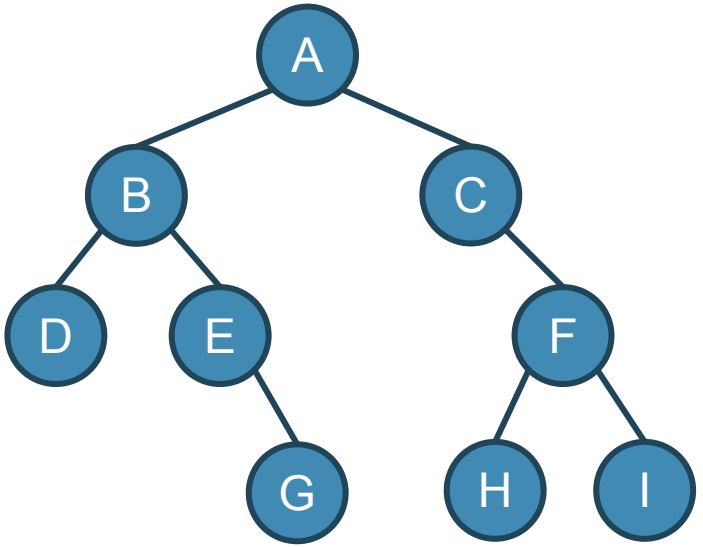
Postorder traversal

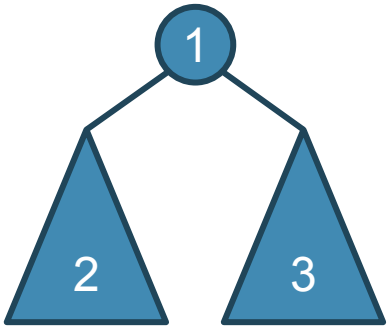


Inorder traversal

Preorder

- **전위순회** Preorder traversal: 노드를 방문하고, 자식들을 순회한다.
  - E.g.) ?

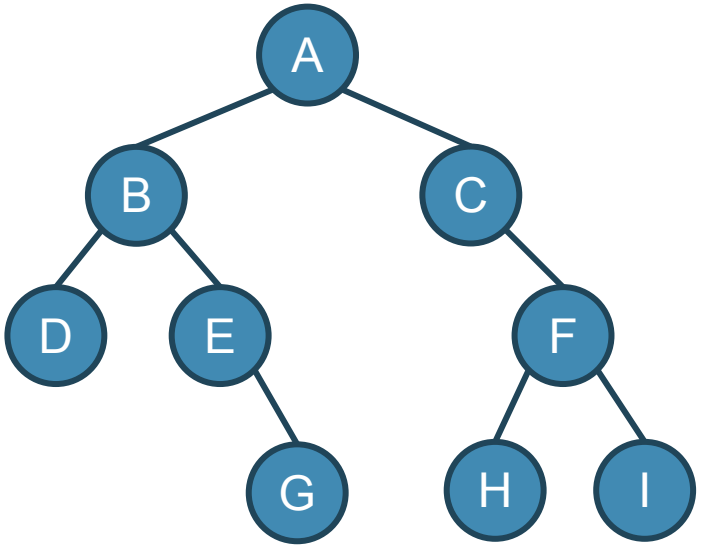


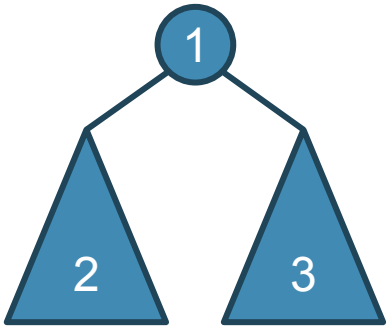


Preorder traversal

Preorder

- **전위순회**Preorder traversal: 노드를 방문하고, 자식들을 순회한다.
  - E.g.) ABDEGCFHI

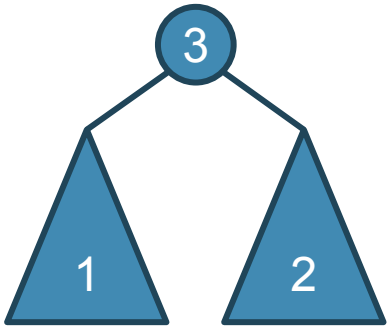




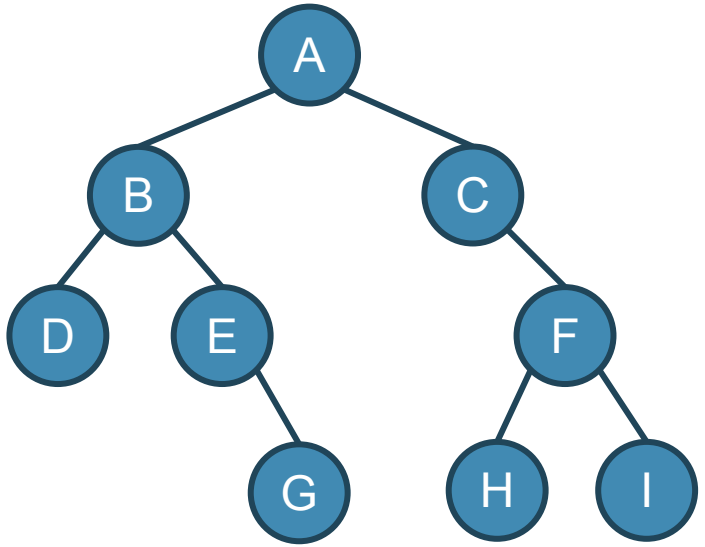
Preorder traversal

Postorder

- **후위순회** Postorder traversal: 자식들을 방문하고, 노드를 방문한다.
  - E.g.) ?

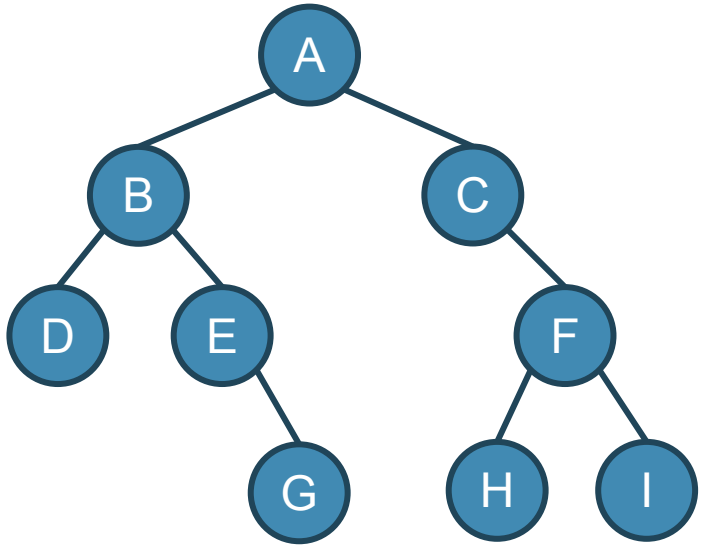


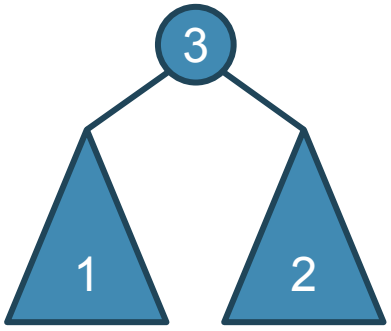
Postorder traversal



Postorder

- **후위순회**Postorder traversal: 자식들을 방문하고, 노드를 방문한다.
  - E.g.) DGEBHIFCA

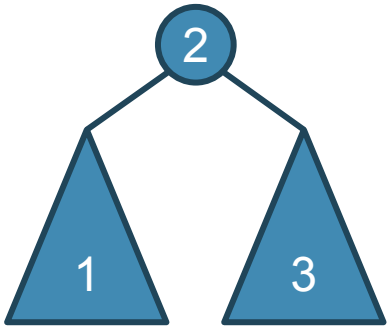




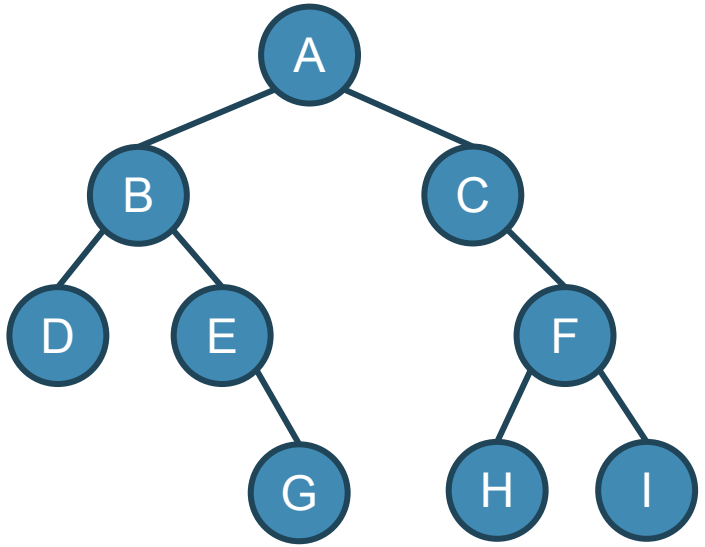
Postorder traversal

Inorder

- **중위순회**Inorder traversal: 왼쪽 subtree를 방문하고, 노드를 방문하고, 오른쪽 subtree를 방문한다.
  - E.g.) ?

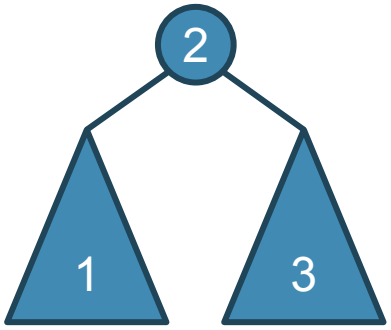


Inorder traversal

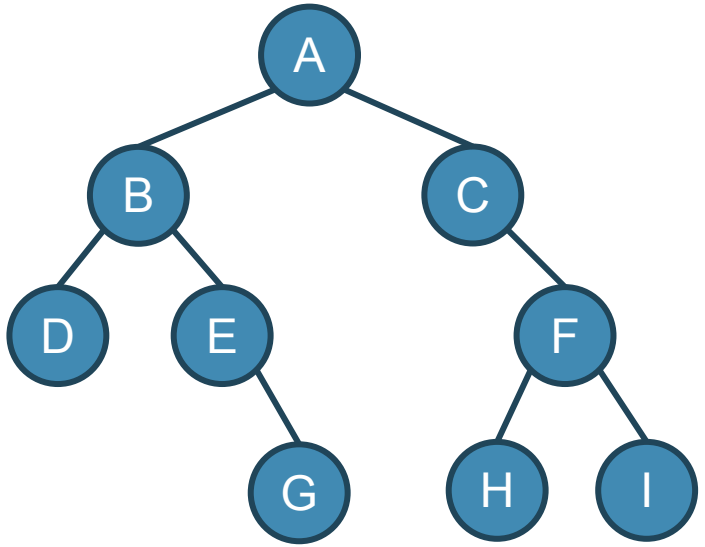


Inorder

- **중위순회**Inorder traversal: 왼쪽 subtree를 방문하고, 노드를 방문하고, 오른쪽 subtree를 방문한다.
  - E.g.) DBEGACHFI



Inorder traversal



Implementing Traversals

- Which implementation is better? why?

```
/** @param rt is the root of the subtree */
void preorder(BinNode rt) {
    if (rt == null) return; // Empty subtree - do nothing
    visit(rt); // Process root node
    preorder(rt.left()); // Process all nodes in left
    preorder(rt.right()); // Process all nodes in right
}


void preorder2(BinNode rt) {
    visit(rt);
    if (rt.left() != null) preorder2(rt.left());
    if (rt.right() != null) preorder2(rt.right());
}
```

What You Need to Know

- The concept of (binary) tree, and its terms
- Idea and proof of full binary tree theorem and its corollary
- How to perform three main traversals for a given tree; how to implement the traversals

Questions?