자료 구조

(tree)

https://youtu.be/lQd4LFnTBZc

송경주

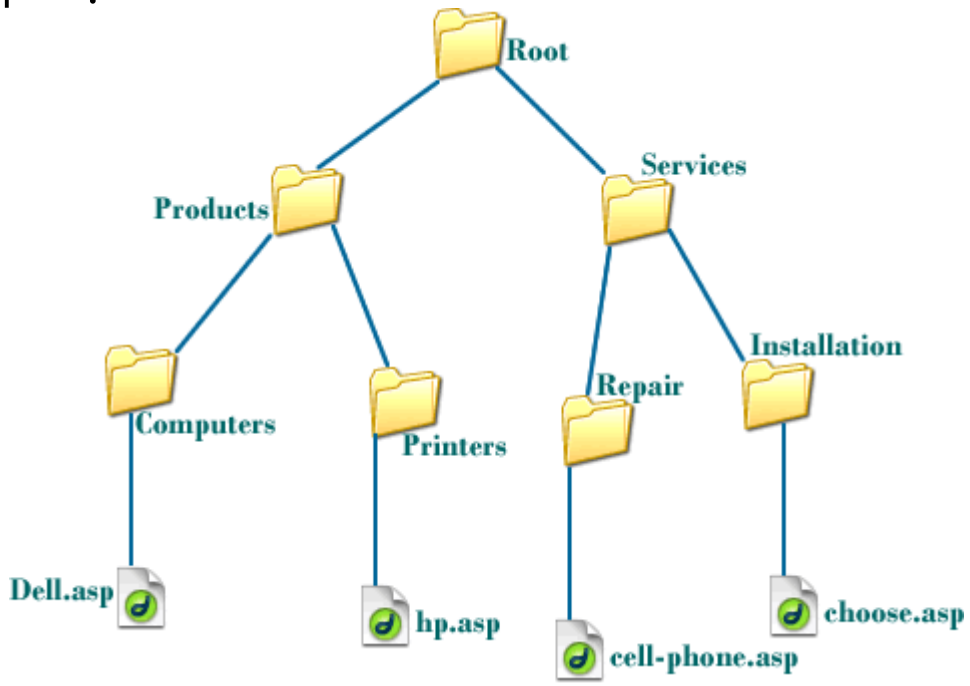
목차

트리 (tree)

- 트리(tree)란?

계층적인 구조를 가진 자료를 표현하는데 적합한 자료구조.

Ex) 컴퓨터 디스크의 디렉터리 구조, 가족구성도 등등



트리 (tree) - 용어정리

노드(node) : 트리의 구성 요소

루트(root)노드 : 부모가 존재하지 않는 최상위 노드

간선(edge) : 루트와 서브 트리를 잇는 연결선

부모 노드(parent node) :

B, C 의 부모 노드 → A

D, E 의 부모 노드 → B

자식 노드(children node) :

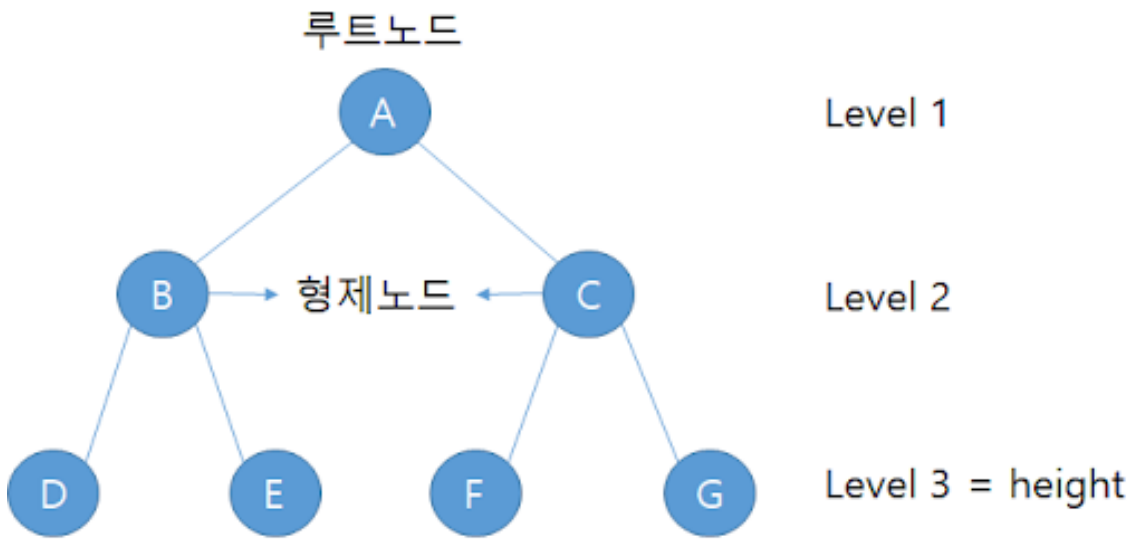
A 의 자식 노드 → B, C

B 의 자식 노드 → D, E

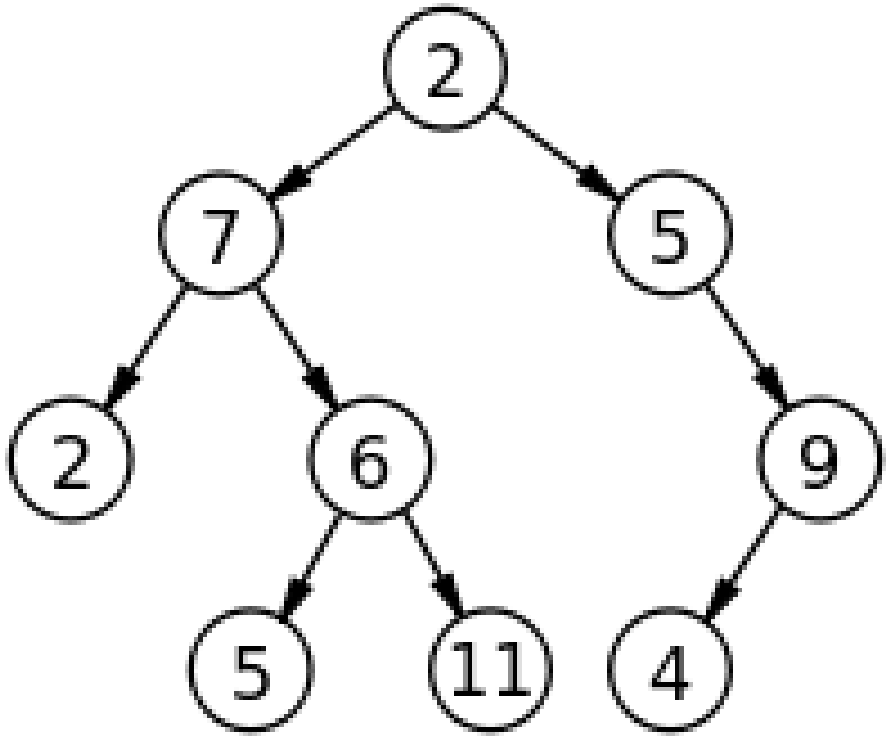
형제 노드(sibling) : B 와 C

단말 노드 : 자식 노드가 없는 노드 → D, E, F, G

높이(height) : 트리가 가지고 있는 최대 레벨



트리 (tree) - 용어정리



- 2는 루트 노드.
- 7은 2와 6의 부모 노드.
- 5는 7의 형제 노드.
- 2와 6은 7의 자식 노드.
- 7의 차수는 '2'이다.
- 트리의 높이는 4 이다.

트리 (tree)

일반 트리

이진 트리

일반 트리 : 자식 노드의 개수가 N개인 트리.

-각 노드마다 링크 필드의 개수가 달라짐

(노드의 크기가 고정되지 않음)

-고정되지 않은 노드의 개수는 프로그램을 복잡하게 만듦.

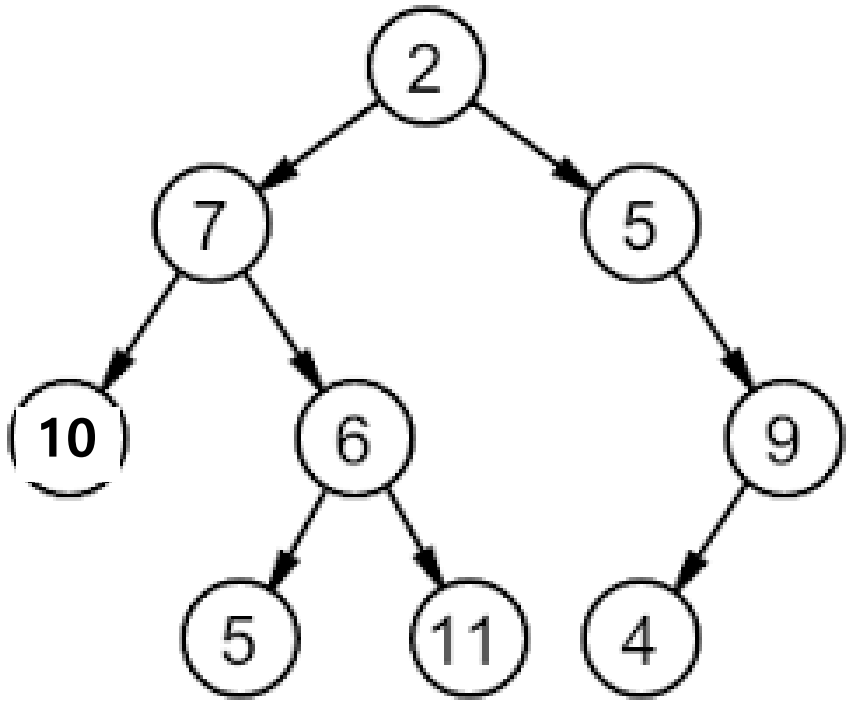
→ 이진 트리로 변환하여 사용 가능

이진 트리 : 자식 노드의 개수가 2개인 트리.

이진 트리 (binary tree)

이진 트리 (binary tree) : 모든 노드가 2개의 서브 트리를 가지고 있는 트리

→서브 트리는 공집합 가능 (즉, 모든 노드의 차수가 2 이하인 트리)



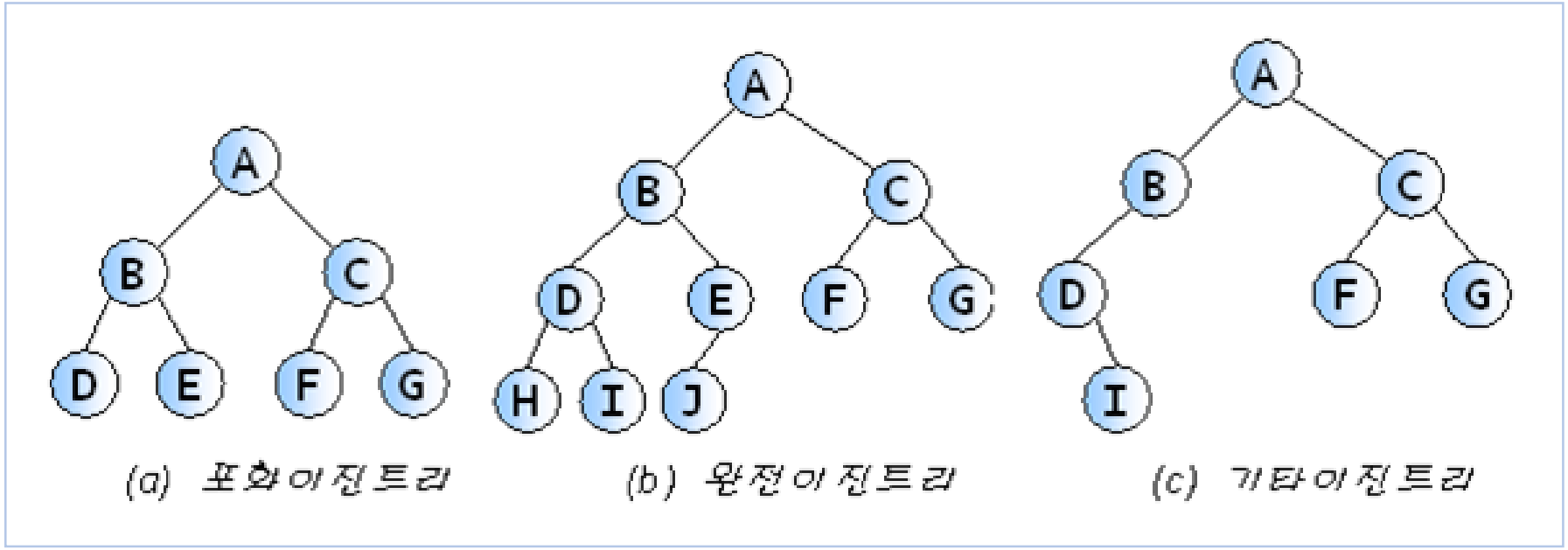
왼쪽의 트리는 이진 트리의 조건을 만족한다.

10은 자식 노드 2개를 공집합으로 가짐.

5의 자식 노드는 공집합과 9.

9의 자식 노드는 공집합과 4.

이진 트리 (binary tree) 분류

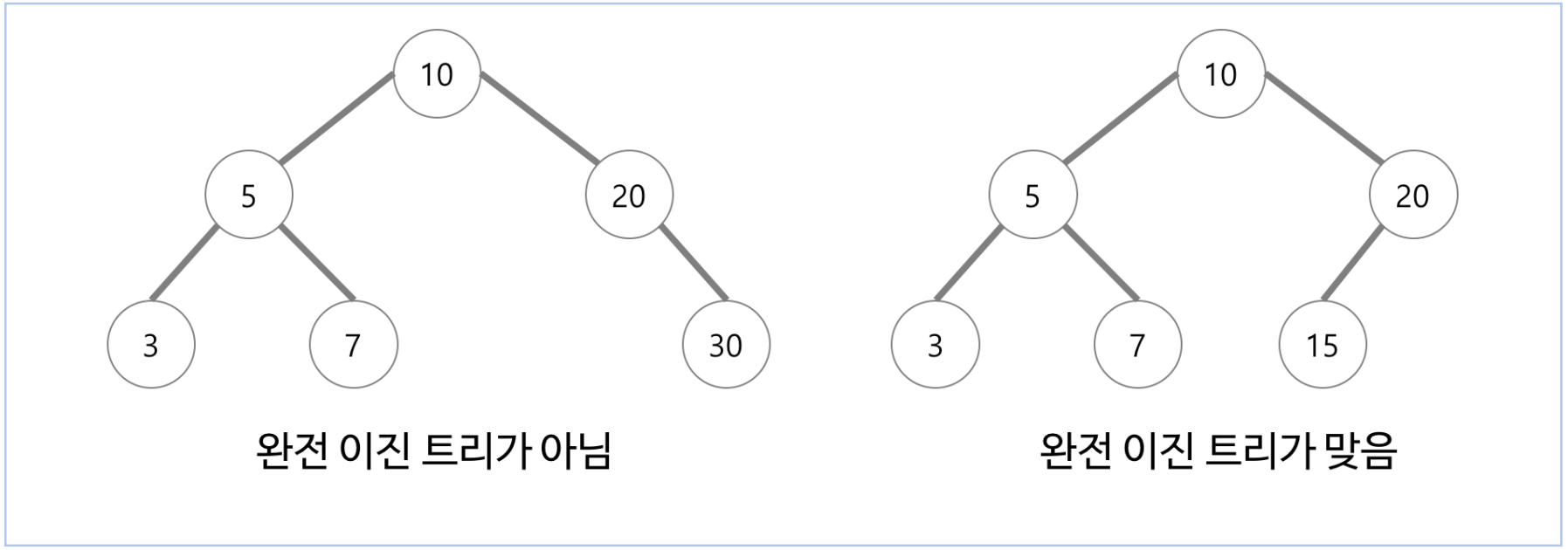


포화 이진 트리(full binary tree) : 트리 각 레벨의 노드가 꽉 차 있는 트리

완전 이진 트리(complete binary tree) : 높이가 k일 때, 레벨 1부터 k-1까지는 노드가 모두 채워져 있고 레벨 k에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져 있는 이진 트리

(마지막 레벨에서는 노드가 꽉 차 있지 않아도 되지만 중간에 빈 곳이 있어서는 안된다)

완전 이진 트리(complete binary tree)



왼쪽의 트리는 마지막 레벨에서 중간이 비어 있으므로

완전 이진 트리가 아니다.

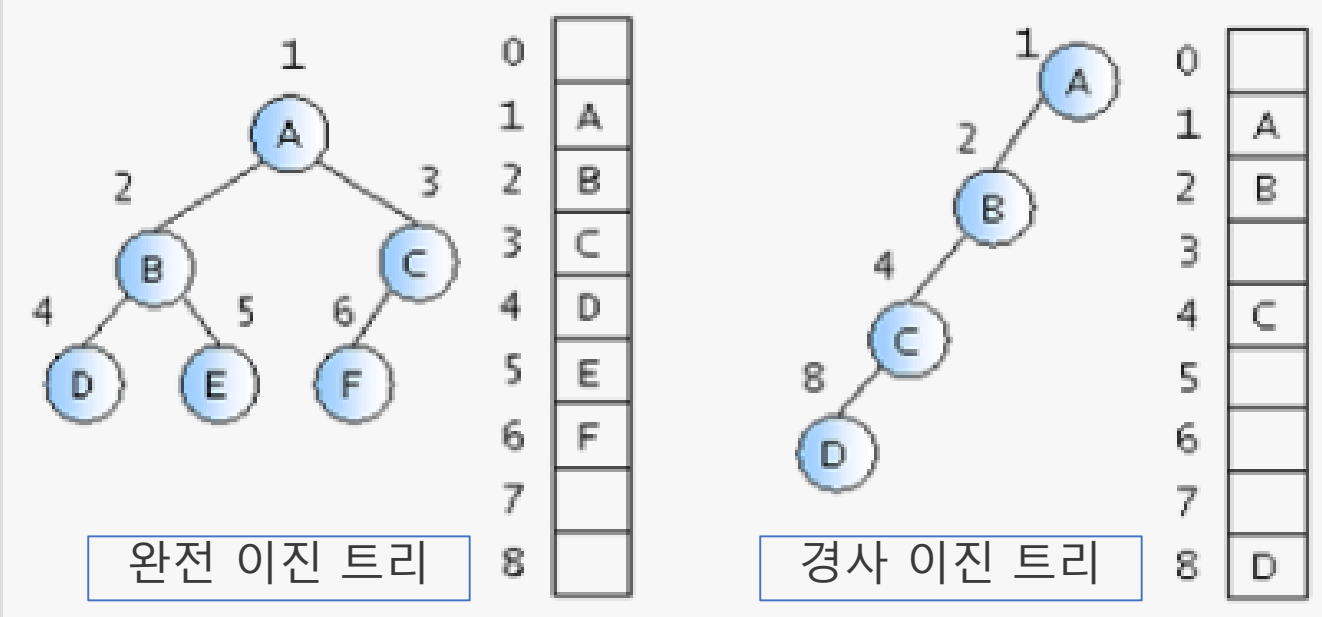
이진 트리(complete binary tree) 표현

1. 배열을 이용하는 방법

2. 포인터를 이용하는 방법

이진 트리 - 배열 표현

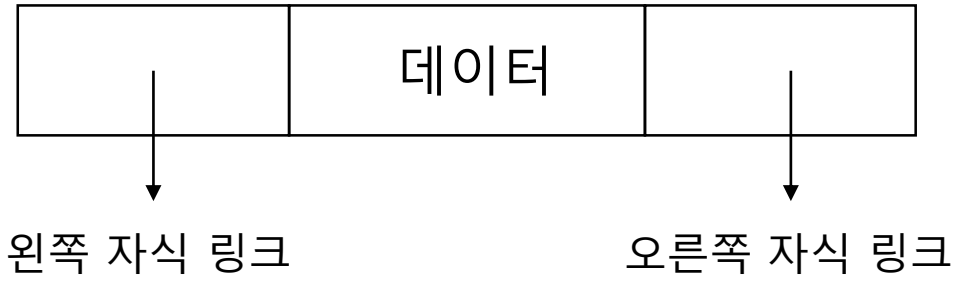
배열 표현법 : 트리의 높이가 k이면 2^k-1 개의 공간을 연속적으로 할당한 후, 완전 이진 트리의 번호대로 노드들을 배열에 저장한다.



1. 노드들은 우선 번호가 매겨지고 이 번호를 따라 배열에 저장된다.
2. A는 노드 번호가 1이므로 배열의 인덱스 1에 저장되고 B는 노드 번호가 2이므로 배열의 인덱스 2에 저장되었다.
3. 경사 이진 트리와 같이 일반 이진 트리인 경우 오른쪽 그림과 같이 저장은 가능하지만 기억 공간의 낭비가 심해진다.

이진 트리 - 링크 표현

링크 표현법 : 노드가 구조체로 표현되고, 각 노드가 포인터를 가지고 있어서
이 포인터를 이용하여 노드와 노드를 연결하는 방법.



이진 트리의 순회

이진 트리 순회(traversal) : 이진 트리에 속하는 모든 노드를 한번씩 방문하여 노드가 가지고 있는 데이터를 목적에 맞게 처리하는 것

전위 순회 (preorder traversal) : VLR → 루트를 서브 트리에 앞서서 먼저 방문

중위 순회 (inorder traversal) : LVR → 루트를 왼쪽과 오른쪽 서브 트리 중간에 방문

후위 순회 (postorder traversal) : LRV → 루트를 왼쪽과 오른쪽 서브 트리 방문 후 방문

(**V** : 트리를 방문하는 작업, **L** : 왼쪽 서브 트리 방문, **R** : 오른쪽 서브 트리 방문)

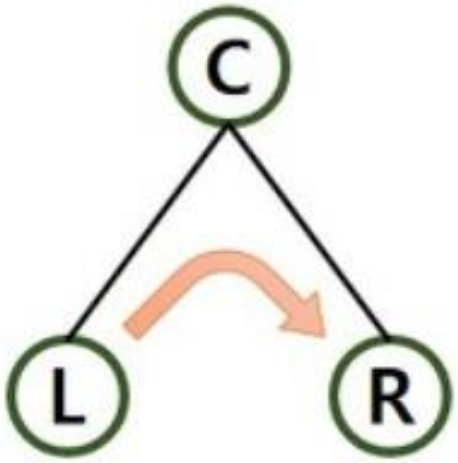
이진 트리의 순회

전위 순회 (preorder traversal) : VLR → 루트를 서브 트리에 앞서서 먼저 방문

1. **루트 노드를 방문 (V)**
2. **왼쪽 서브 트리 방문 (L)**
3. **오른쪽 서브 트리 방문 (R)**

이진 트리의 순회

중위 순회 (inorder traversal) : LVR → 루트를 왼쪽과 오른쪽 서브 트리 중간에 방문

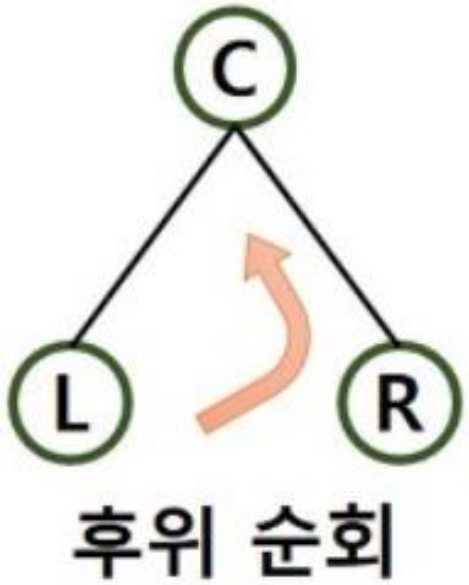


중위 순회

1. **왼쪽 서브 트리 방문 (L)**
2. **루트 노드 방문 (V)**
3. **오른쪽 서브 트리 방문 (R)**

이진 트리의 순회

후위 순회 (postorder traversal) : LRV → 루트를 왼쪽과 오른쪽 서브 트리 방문 후 방문



1. **왼쪽 서브 트리 방문 (L)**
2. **오른쪽 서브 트리 방문 ®**
3. **루트 노드 방문 (V)**

순회 방법 선택

전위 순회? 후위 순회? 후위 순회?
어떤 방법을 선택해야 할까?

순서의 상관 없이 모든 노드를 전부 방문 → **3가지 방법 상관 X**
자식 노드를 먼저 처리한 다음에 부모 노드를 처리해야 하는 경우 → **후위 순회**
부모 노드를 먼저 처리한 다음에 자식 노드를 처리해야 하는 경우 → **전위 순회**

이진 탐색 트리

이진 탐색 트리(binary search tree) : 이진 트리 기반의 탐색을 위한 자료 구조, 이진 탐색 트리의 성질을 만족하는 이진 트리

[이진 탐색 트리의 성질]

1. 모든 노드의 키는 유일하다.
2. 왼쪽 서브 트리의 키들은 루트의 키보다 작다.
3. 오른쪽 서브 트리의 키들은 루트의 키보다 크다.
4. 왼쪽과 오른쪽 서브 트리도 이진 탐색 트리이다.

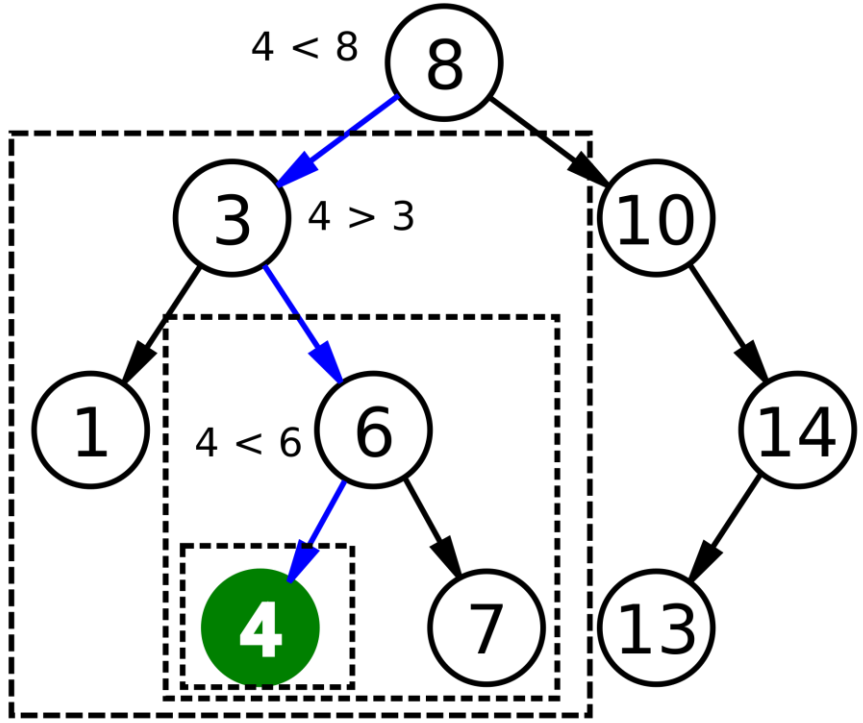
이진 탐색 트리-탐색

[탐색 연산]

1. 비교한 결과가 **같으면** 탐색 종료.

2. 주어진 키 값이 루트 노드의 키 값보다 **작으면** 탐색은 이 루트 노드의 **왼쪽 자식**을 기준으로 다시 시작.

3. 주어진 키 값이 루트 노드의 키 값보다 **크면** 탐색은 이 루트 노드의 **오른쪽 자식**을 기준으로 다시 시작.

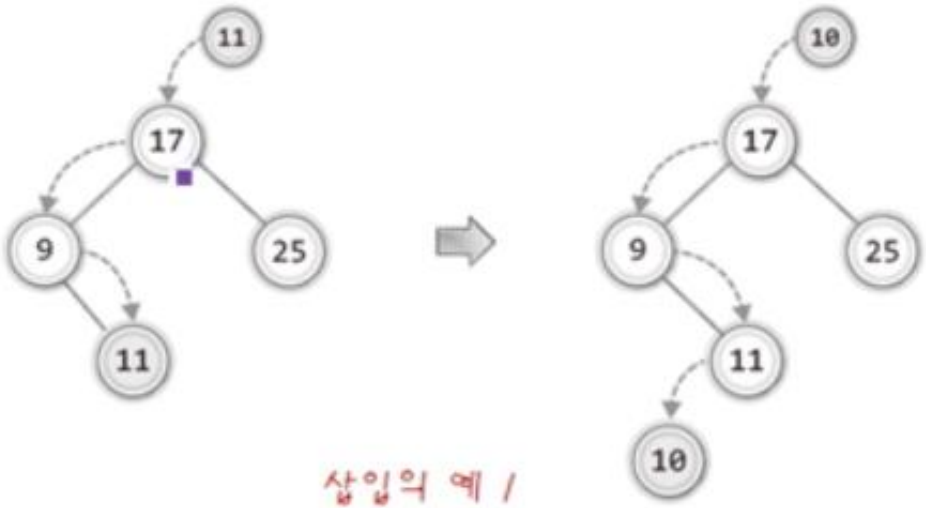


이진 탐색 트리-삽입

[삽입 연산]

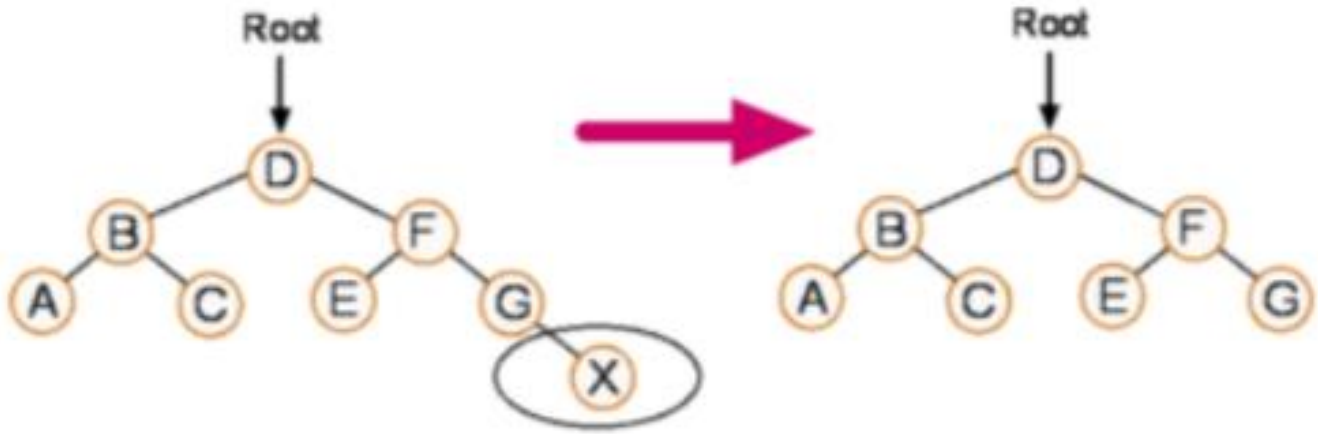
1. **삽입 하기 전 검색 수행**

2. **트리를 검색한 후 키와 일치하는 노드가 없으면 마지막 노드에서 키와 노드의 크기를 비교하여 왼쪽이나 오른쪽에 새로운 노드 삽입.**



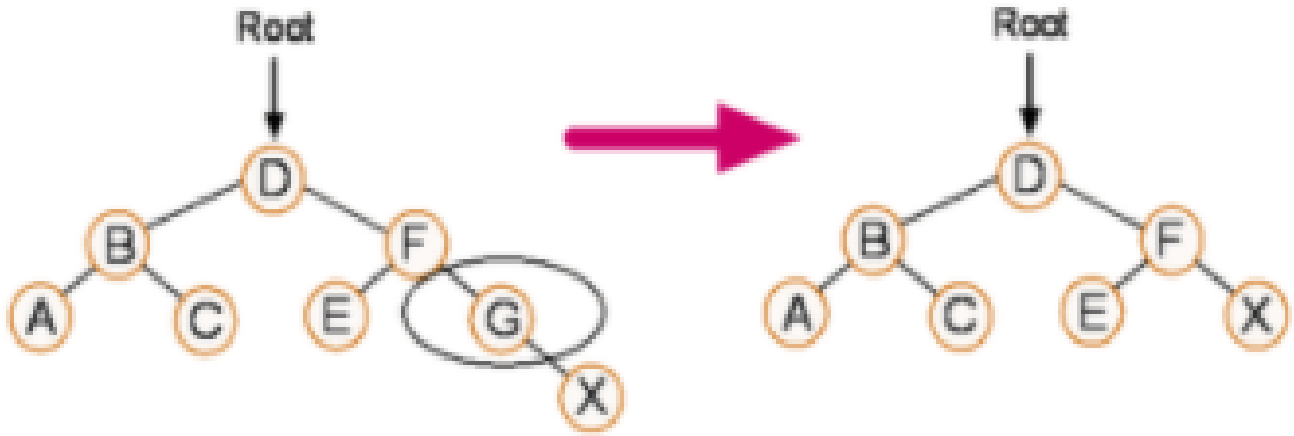
이진 탐색 트리-삭제

1. **단말 노드 삭제 :** 해당 노드를 단순히 삭제



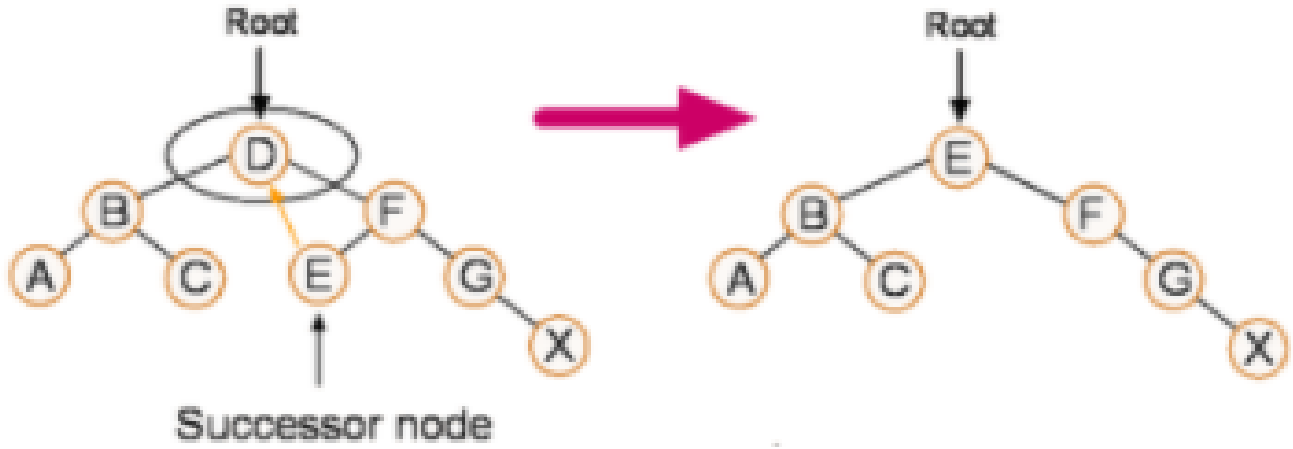
이진 탐색 트리-삭제

2. 자식 노드가 1개인 노드 삭제 : 해당 노드를 삭제하고 그 위치에 해당 노드의 자식 노드를 대입



이진 탐색 트리-삭제

3. 자식 노드가 2개인 노드 삭제 : 삭제하고자 하는 노드의 값을 해당 노드의 왼쪽 서브 트리에서 가장 큰 값으로 대체하거나 오른쪽 서브 트리에서 가장 작은 값으로 대체한 뒤, 해당 노드(왼쪽 서브 트리에서 가장 큰 값을 가지는 노드 또는 오른쪽 서브 트리에서 가장 작은 값을 가지는 노드)를 삭제한다.



감사합니다 :-)