

# 자료구조와 알고리즘

10강 - 트리 자료구조

LECTURED BY SOONGU HONG



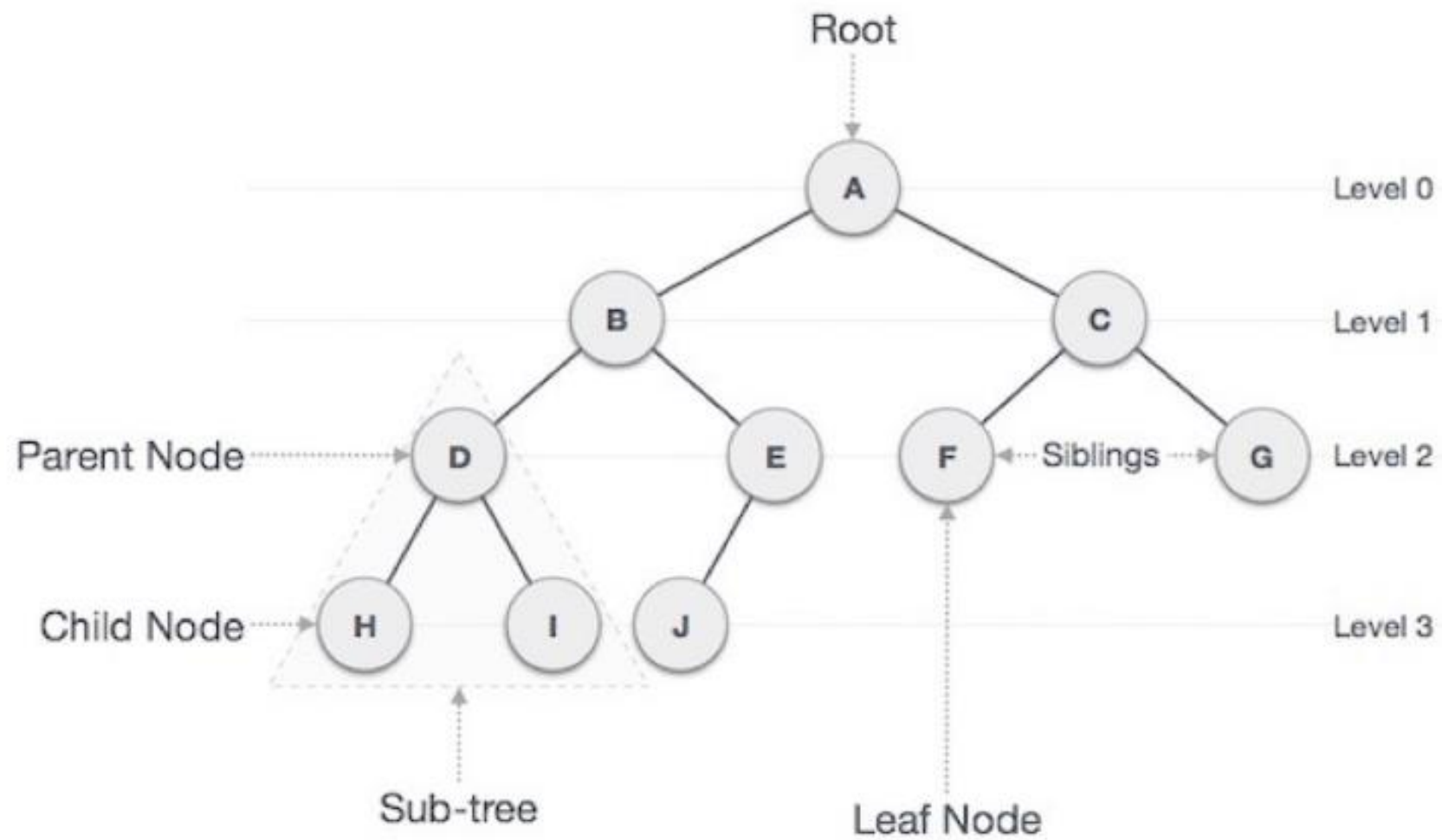
# 1. 트리 (TREE)

## \* 왜 트리를 사용하는가?

- 우리가 지금까지 다루어온 자료구조들은 선형적 구조(linear structure)였습니다.
- 따라서 선형적 자료구조가 갖는 단점들을 가지고 있을 수 밖에 없습니다.
- 예를 들면 정렬된 배열의 경우 탐색에 있어서는 빠르지만( $O(\log N)$ ), 삽입이나 삭제에 대해서는 상당히 느린 성능( $O(N)$ )을 보여줍니다.
- 반대로 연결리스트는 삽입, 삭제에서는 빠르지만 ( $O(\log N)$ ), 탐색은 느린 성능( $O(N)$ )을 보여줍니다.
- 그러나 **트리는 비선형 자료구조**(non-linear structure)로 삽입, 삭제, 탐색 모두에서 빠른 성능 ( $O(\log N)$ )을 가지고 있습니다.

## \* 트리 관련 용어

- 경로(path): 어떤 한 노드에서 다른 노드까지 링크를 통해 이동했을 때, 거쳐온 일련의 노드의 집합을 경로라 한다.
- 루트(root) : 트리의 가장 상위에 있는 노드, 루트는 언제나 하나만 존재해야 한다.
- 부모, 자식 : 링크로 연결되어 있는 노드 중 위에 있는 노드를 부모 노드, 아래에 있는 노드를 자식 노드라 한다.
- 잎(leaf) : 자식을 가지고 있지 않은 노드를 잎 노드라 한다. 단말노드라고도 부름.
- 키(key) : 키란 자료 항목을 찾거나 다른 동작을 하기 위해 필요한 값으로, 각 자료 항목들을 구분해주는 역할을 하는 값이다.
- 하위트리(subtree): 하나의 큰 트리에 속해 있는 부분을 하위 트리라고 한다.
- 방문(visiting) : 노드에 도착해 자료 값을 읽는 것.
- 순회(traversing) : 트리 노드 전체를 방문하는 것
- 레벨(level) : 루트를 0레벨이라 하면 그 자식은 1레벨, 그의 자식은 2레벨이라 함.



## \* 트리의 일반적 특징

### 1. 트리구조에서 한 노드에서 다른 노드로 가는 경로는 유일하다.

- 트리구조에서 임의의 두 노드를 선택했을 경우, 이 두 노드는 최소 공통 선조를 갖는다.
- 최소 공통 선조란 임의의 두 노드가 가질 수 있는 공통 선조들 중에서 가장 가까운 선조를 말한다.
- 트리를 타는 경로가 중복됨이 없고, 되돌아감이 없다면 두 노드 간의 경로는 반드시 한 노드에서 최소 공통 선조까지 올라갔다가 다시 다른 노드로 내려오는 유일한 경로만이 존재한다.
- 이 성질은 그래프 구조와 트리 구조를 구분하는 아주 중요한 성질이다.

### 2. N개의 노드를 갖는 트리는 N-1개의 링크를 가진다.

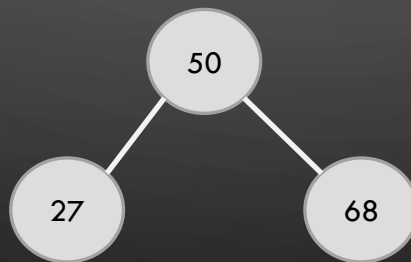
- 트리는 그래프와 달리 루트를 제외하고는 모든 노드가 자신의 선조를 향한 단 하나의 링크를 갖고 있다.

A decorative graphic on the left side of the slide, consisting of white lines and circles on a dark gray background. The lines are vertical and horizontal, with some branching out, resembling a circuit board or a stylized tree structure. The circles are small and are placed at various points along the lines.

## 2. 이진 트리

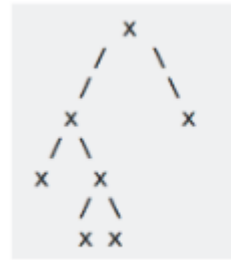
## \* 이진 트리

- 트리 구조 중 자식을 최대로 둘까지만 가질 수 있는 트리구조를 이진트리(binary tree)라 한다.
- 이진트리는 여러 트리구조 중 가장 간단하면서 보편적인 구조이며, 이진탐색트리라고도 한다.
- 이진트리에서는 각 노드들은 자식이 없거나, 하나 또는 두개의 자식 노드를 가질 수 있다.
- 부모의 왼쪽에 있는 자식노드를 왼쪽 자식(left child), 오른쪽을 오른쪽 자식(right child)라 부른다.
- 이진트리의 중요한 특성 중 하나는 왼쪽 자식의 키(key)는 부모노드의 키보다 작고, 오른쪽 자식의 키는 부모노드의 키보다 크다는 것이다.

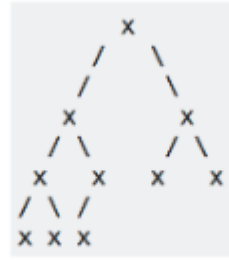




\* **꼭 찬(FULL) 이진트리 VS 완전(COMPLETE) 이진트리 VS 포화(PERFECT) 이진트리**



Full



Complete

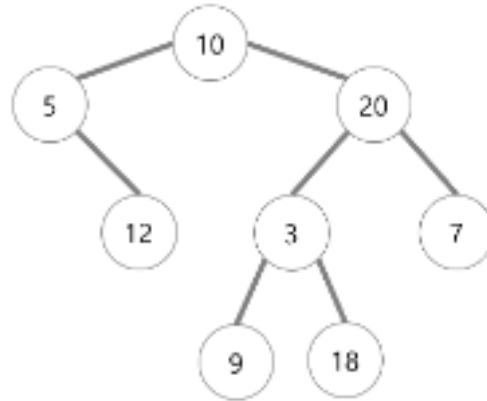


Perfect

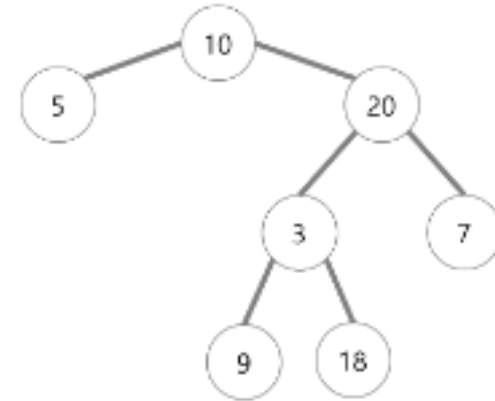
## \* 짝 찬(FULL) 이진트리

- 전 이진 트리(Full Binary Tree 또는 Strictly Binary Tree)

◦



전 이진 트리가아님



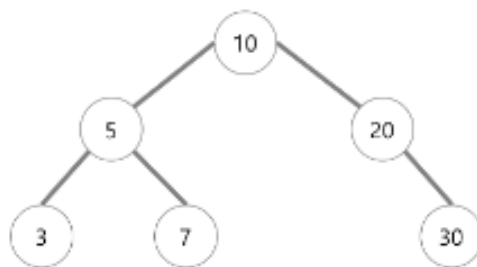
전 이진 트리가맞음

- 모든 노드가 0개 또는 2개의 자식 노드를 갖는 트리.

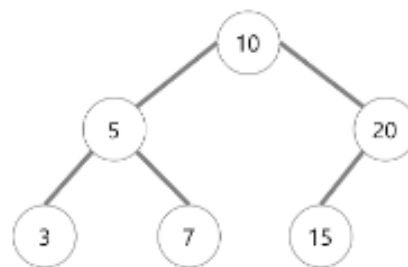
## \* 완전(COMPLETE) 이진트리

- 완전 이진 트리(Complete Binary Tree)

- 



완전 이진 트리가아님



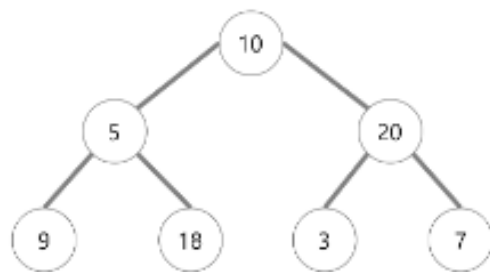
완전 이진 트리가맞음

- 트리의 모든 높이에서 노드가 꽉 차 있는 이진 트리. 즉, 마지막 레벨을 제외하고 모든 레벨이 완전히 채워져 있다.
  - 마지막 레벨은 꽉 차 있지 않아도 되지만 노드가 왼쪽에서 오른쪽으로 채워져야 한다.
  - 마지막 레벨  $h$ 에서  $(1 \sim 2^{h-1})$ 개의 노드를 가질 수 있다.
  - 또 다른 정의는 가장 오른쪽의 잎 노드가 (아마도 모두) 제거된 포화 이진 트리다.
  - 완전 이진 트리는 배열을 사용해 효율적으로 표현 가능하다.

## \* 포화(PERFECT) 이진트리

- 포화 이진 트리(Perfect Binary Tree)

- 



포화 이진 트리

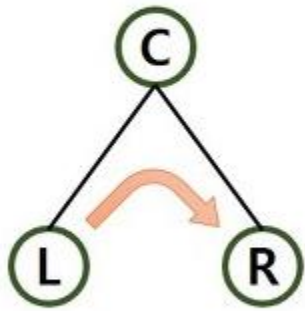
- 전 이진 트리이면서 완전 이진 트리인 경우
    - 모든 말단 노드는 같은 높이에 있어야 하며, 마지막 단계에서 노드의 개수가 최대가 되어야 한다.
    - 모든 내부 노드가 두 개의 자식 노드를 가진다.
    - 모든 말단 노드가 동일한 깊이 또는 레벨을 갖는다.

A decorative graphic on the left side of the slide, consisting of white lines and circles on a dark gray background, resembling a circuit board or a stylized tree structure.

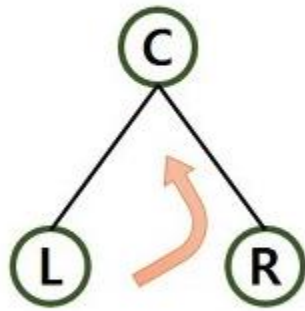
# 3. 순회

## \* 순회의 종류

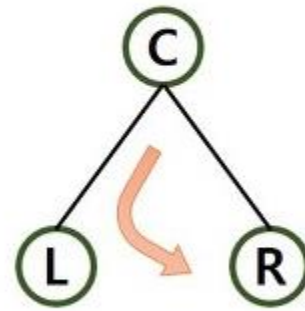
C: Center  
L: Left  
R: Right



중위 순회

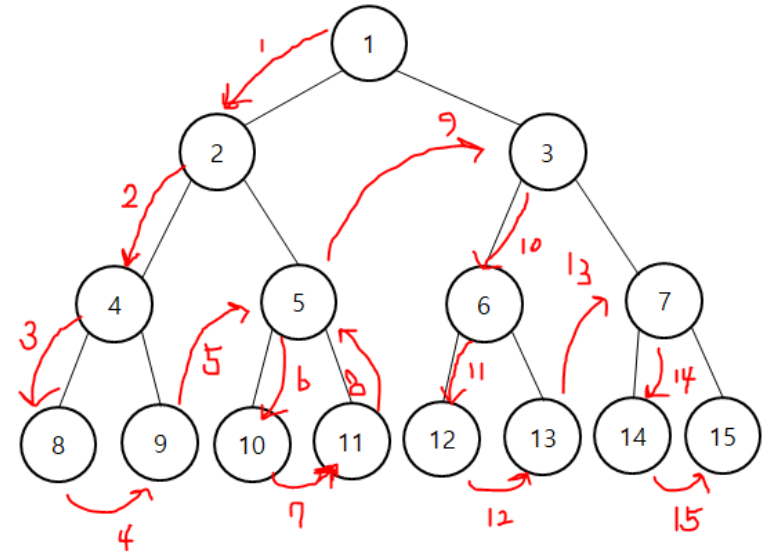


후위 순회



전위 순회

# \* 전위 순회 (PREORDER)



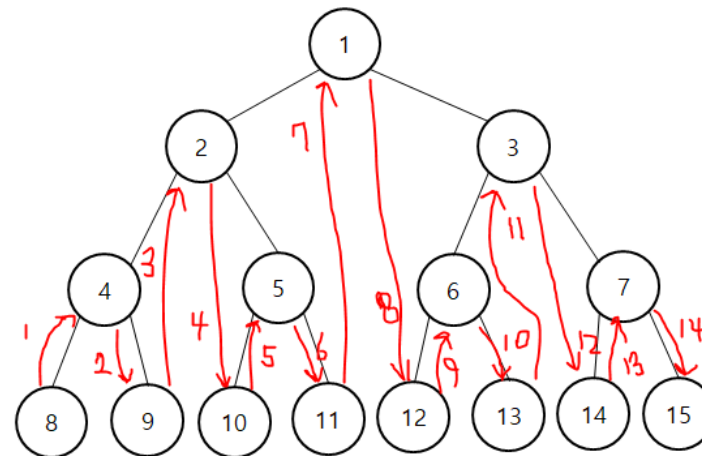
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15

전위 순회 Preorder Traversal

root -> left -> right

부모노드 -> 왼쪽 자식 노드 -> 오른쪽 자식 노드

# \* 중위 순회 (INORDER)



8 4 9 2 10 5 11 1 12 6 13 3 14 7 15

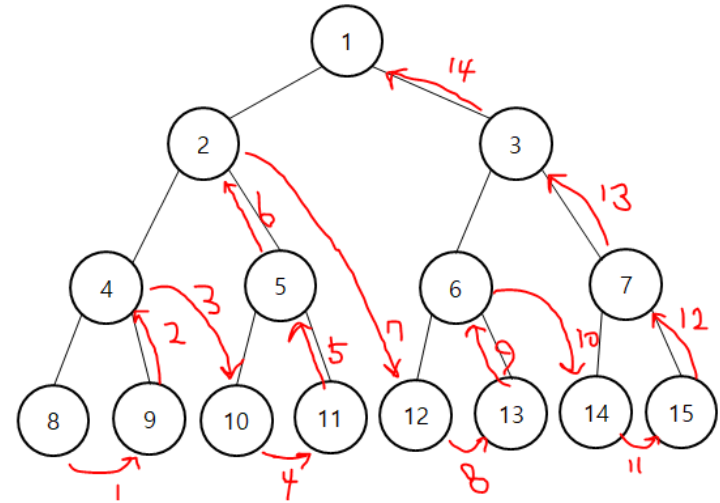
중위 순회 Inorder Traversal

left -> root -> right

왼쪽 자식 노드 -> 부모노드 -> 오른쪽 자식 노드



# \* 후위 순회 (POSTORDER)



8 9 4 10 11 5 2 12 13 6 14 15 7 3 1

후위 순회 Postorder Traversal

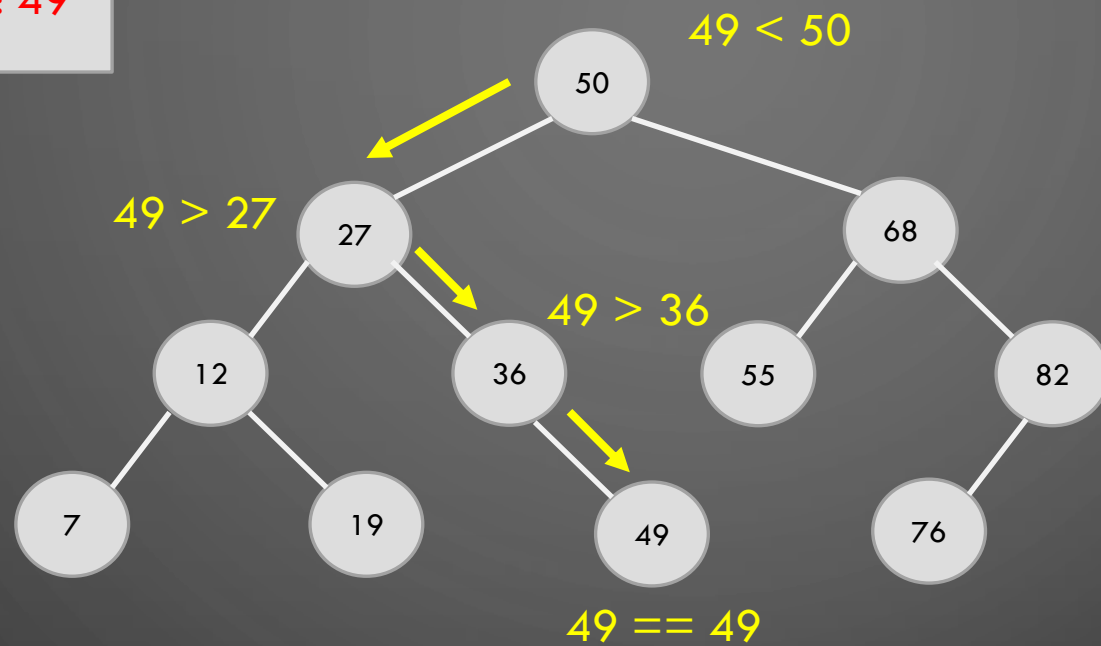
left -> right -> root

왼쪽 자식 노드 -> 오른쪽 자식 노드 -> 부모노드



## 4. 탐색

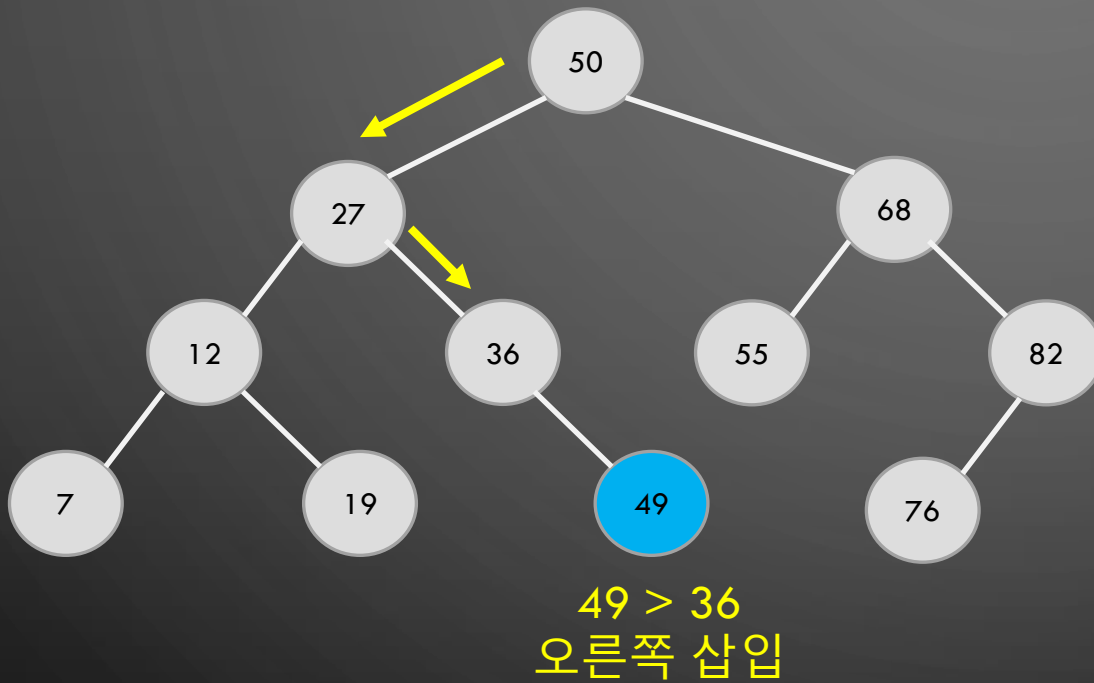
찾는 값: 49



A decorative graphic on the left side of the slide, consisting of white lines and circles on a dark gray background, resembling a circuit board or a stylized tree structure.

## 5. 삽입

삽입 값: 49





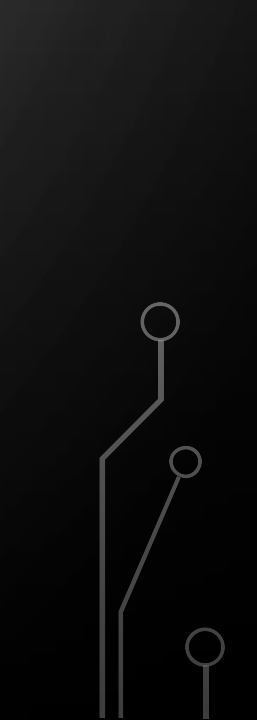
- 삽입과정 수행 시 가장 먼저 해야 할 일은 노드가 삽입될 위치를 찾는 것입니다.
- 노드가 삽입될 위치를 결정하기 위해 적절한 경로를 따라 내려간 뒤 그 위치의 부모가 되는 노드를 찾습니다.
- 이 부모 노드의 키보다 삽입될 노드의 키가 작으면 왼쪽 자식으로, 크다면 오른쪽 자식으로 삽입합니다.

A decorative graphic on the left side of the slide, consisting of a network of white lines and small circles on a dark gray background, resembling a circuit board or a neural network structure.

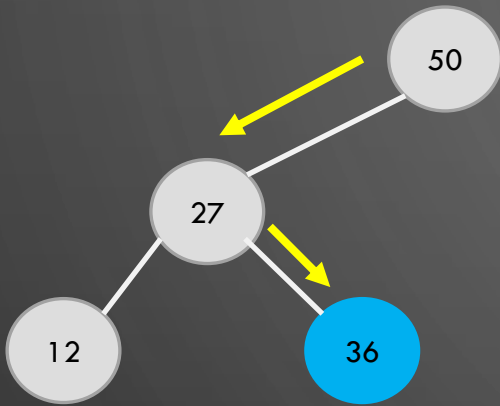
## 6. 삭제



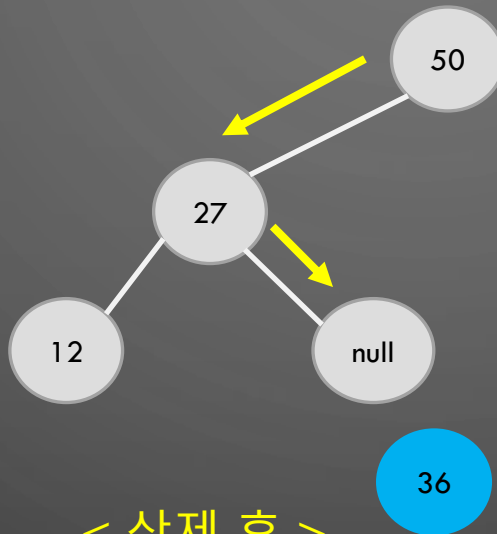
## \* 노드 삭제 상황

1. 삭제하려는 노드가 잎 노드일 때 (즉, 자식노드가 없을 경우)
  2. 삭제하려는 노드의 자식이 하나일 때
  3. 삭제하려는 노드의 자식이 둘일 때
- 
- 
- 

삭제 값: 36



< 삭제 전 >



< 삭제 후 >

## • 상황1: 삭제하려는 노드가 잎 노드

- 삭제 노드의 부모노드를 찾아 삭제 노드를 가리키는 링크를 null로 만들면 된다.
- 만약 삭제 대상이 루트노드라면 루트를 null로 만들어 트리를 삭제한다.



삭제 값: 82

< 삭제 전 >

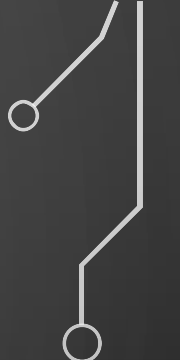
< 삭제 후 >

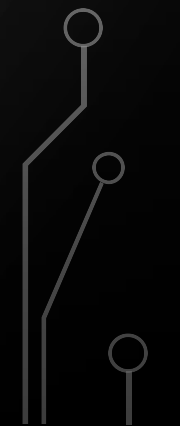
## • 상황2: 삭제하려는 노드의 자식이 하나일 때

- 삭제하려는 노드의 자식노드와 부모노드를 바로 이어준다.
- 만약 삭제 대상이 루트노드라면 그 자식을 새로운 루트노드로 만들어준다.



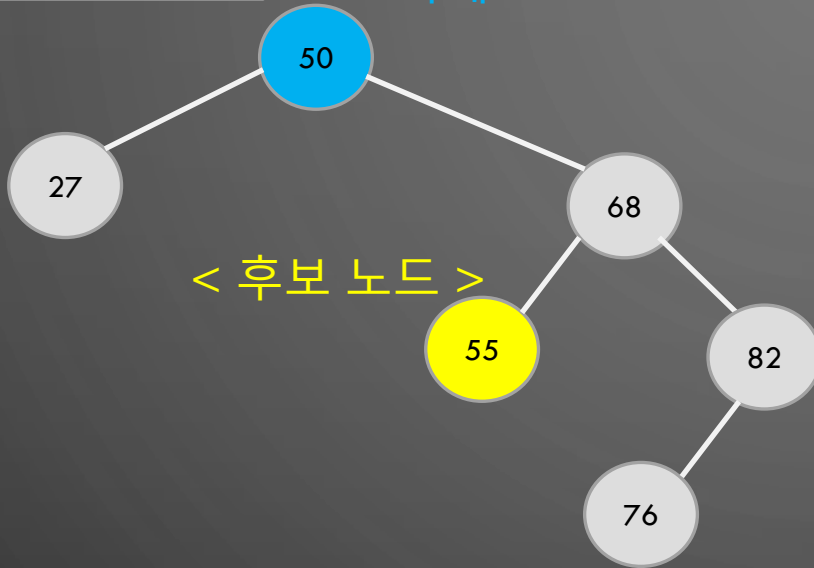
### • 상황3: 삭제하려는 노드의 자식이 둘일 때

- 이 상황은 이전의 두 상황과는 다르게 복잡합니다. 삭제하려는 노드의 자식 중 하나로 그 위치를 대체할 수 없기 때문입니다.
  - 이 상황을 해결하기 위해 가장 먼저해야 하는 일은 삭제될 노드의 위치를 채워줄 올바른 후보노드를 찾는 것입니다.
  - 삭제될 노드보다 큰 키값을 가진 노드 중 가장 작은 키 값을 갖는 노드가 후보노드로 선정됩니다.
- 



삭제 값: 50

< 삭제 노드 >



< 후보 노드 >

## • 삭제노드를 대체할 후보노드 찾기

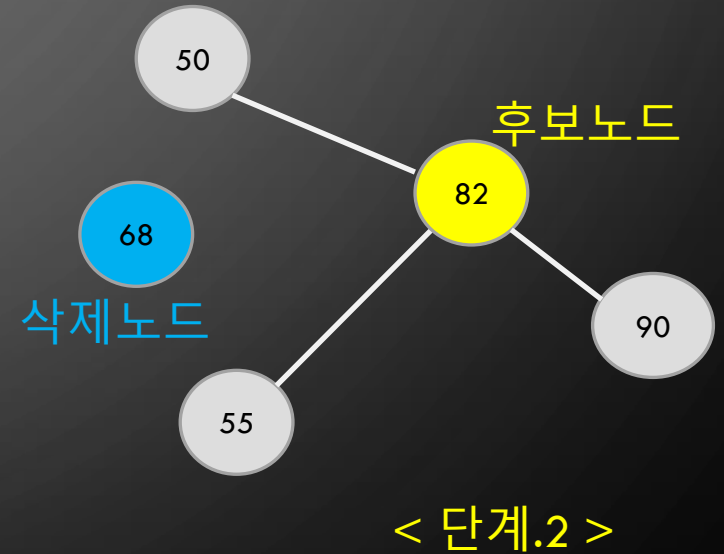
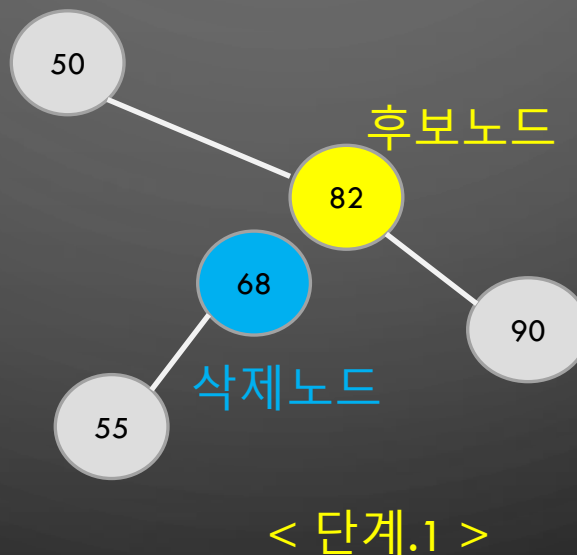
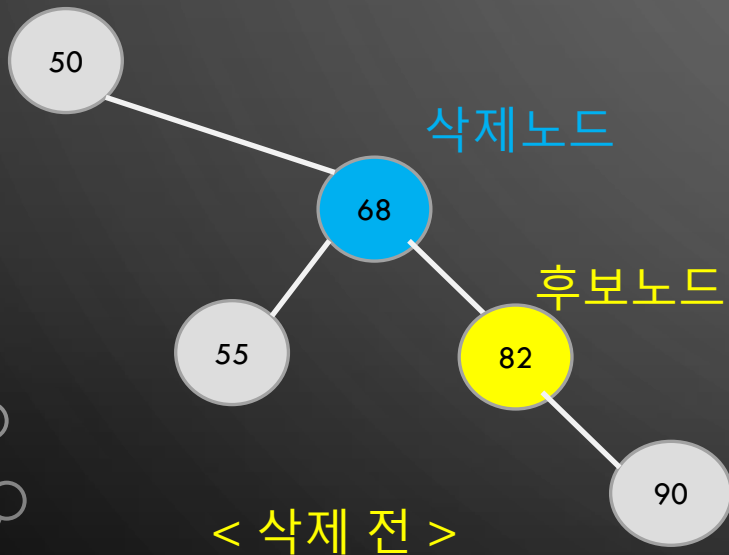
- 삭제될 노드보다 큰 값을 가진 노드 중  
(68, 55, 82, 76)
- 가장 최소값(55) 노드가 후보가된다.

### • 상황3-1: 후보노드가 삭제노드의 오른쪽 자식일 경우

단계 1. 부모노드에서 삭제노드에 대한 링크를 끊고 후보노드로 그 링크를 연결.

단계 2. 삭제노드의 왼쪽 자식은 삭제노드와의 링크를 끊고 후보노드의 왼쪽 자식으로 링크를 연결.

삭제 값: 68



### • 상황3-2: 후보노드가 삭제노드의 왼쪽 자식일 경우

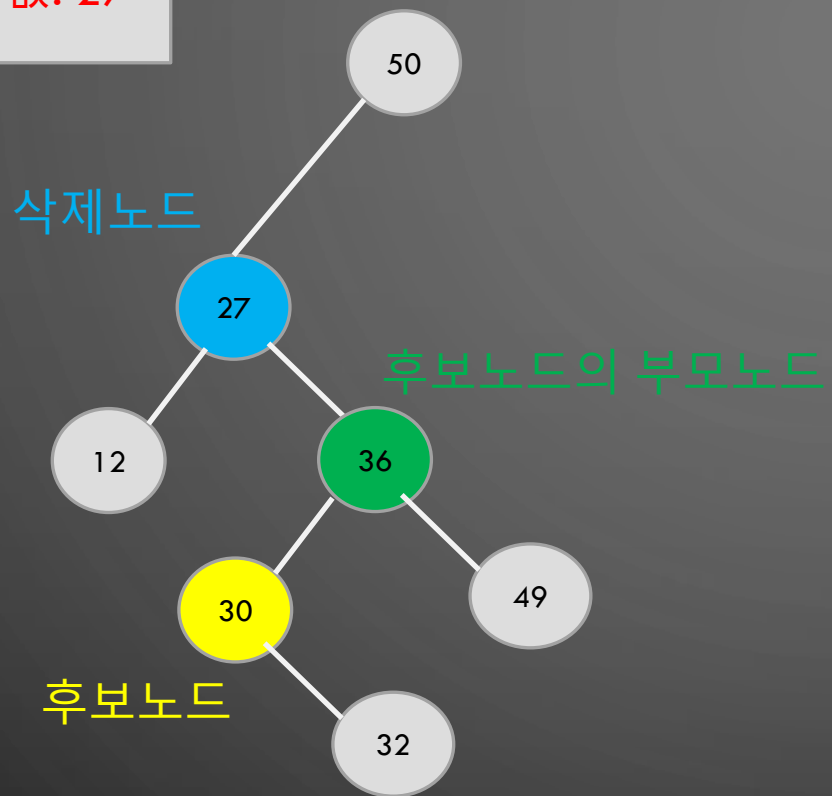
단계 1. 후보노드의 오른쪽 자식을 후보노드의 부모노드의 왼쪽 자식으로 만든다.

단계 2. 삭제노드의 오른쪽 자식을 후보노드의 오른쪽 자식으로 만든다.

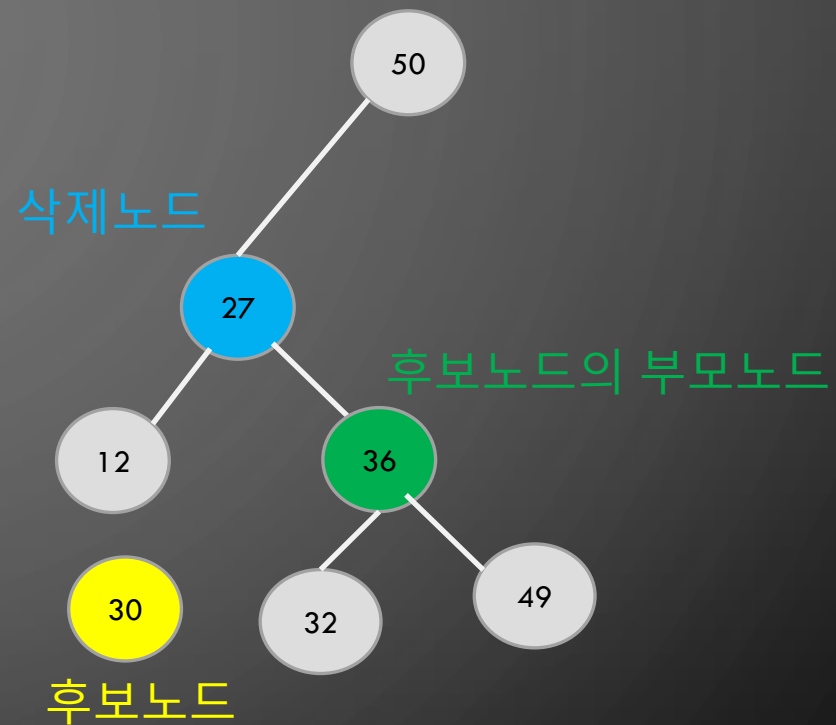
단계 3. 부모노드에서 삭제노드에 대한 링크를 끊고 후보노드로 그 링크를 연결.

단계 4. 삭제노드의 왼쪽 자식은 삭제노드와의 링크를 끊고 후보노드의  
왼쪽 자식으로 링크를 연결.

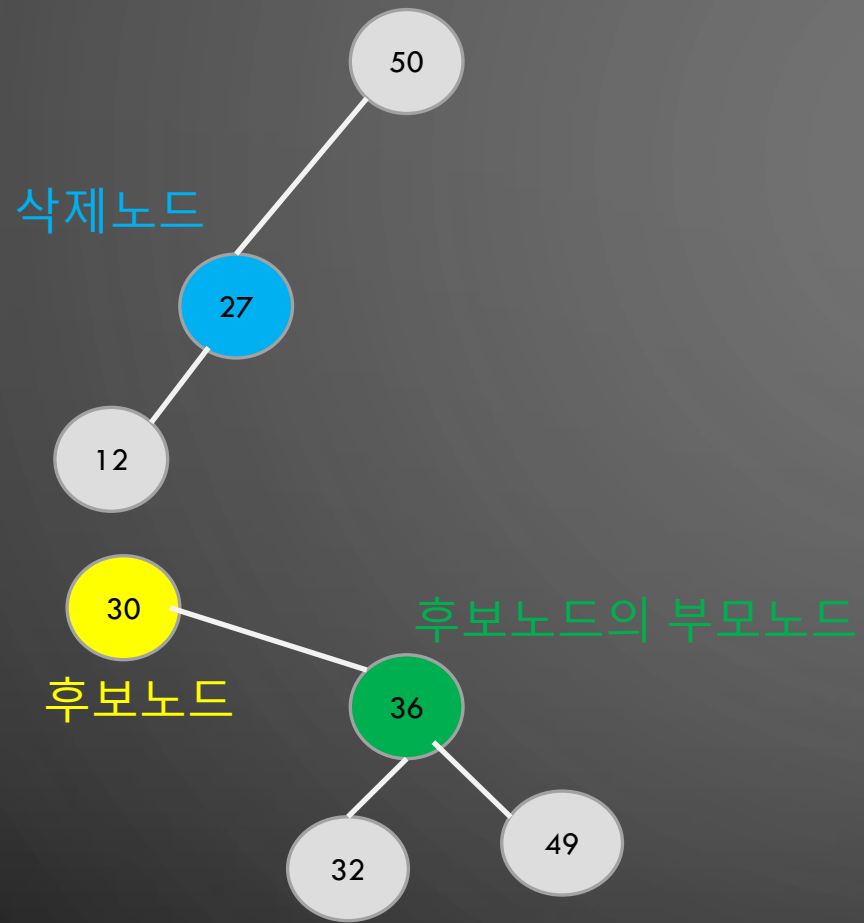
삭제 값: 27



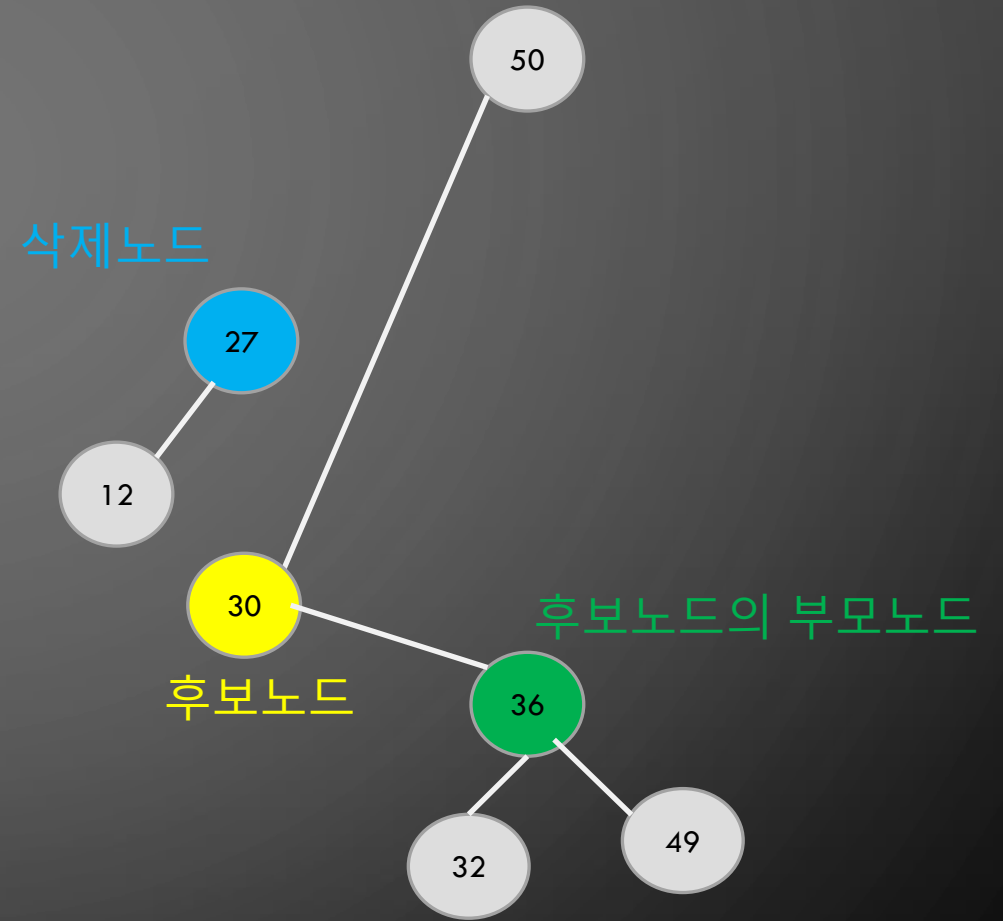
< 삭제 전 >



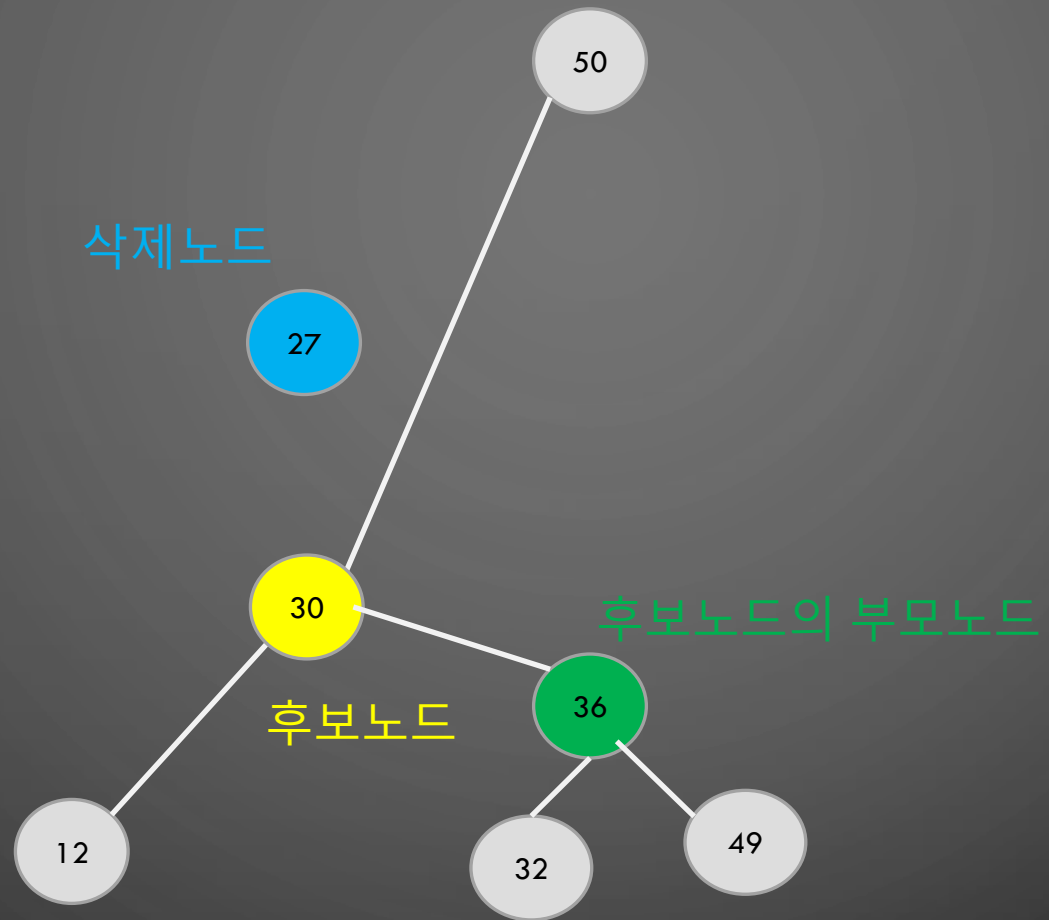
< 단계 1 >



< 단계 2 >



< 단계 3 >



< 단계 4 >