



Algorithm

탐색 알고리즘

2025-04-18

조윤실



목 차



■ 트리 기반 탐색

- 1) 트리 탐색 개요
- 2) 이진 트리
- 3) 트리 순회
- 4) 응용 : 수식 트리
- 5) 이진 탐색 트리: BST

■ 그래프 탐색

- 1) 깊이 우선 탐색(DFS)
- 2) 너비 우선 탐색(BFS)

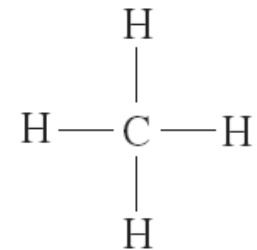
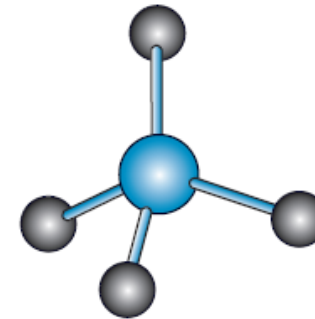
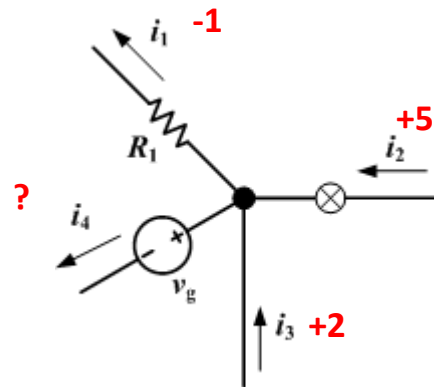
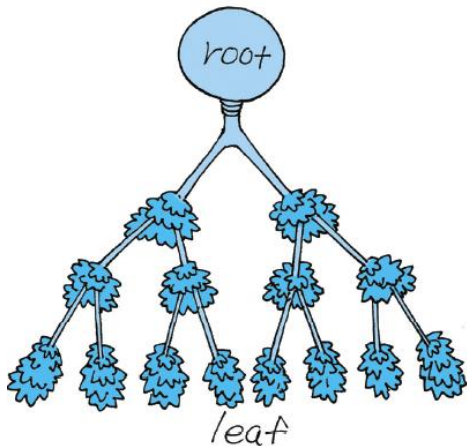
※ 가천대학교 컴퓨터공학과 '알고리즘' 과정

트리 기반 탐색

트리 탐색 개요

트리

- 트리(Tree)는 나무 모양의 자료구조
 - '트리'라는 계층적 자료의 표현에 매우 유용한 구조
 - 트리는 그래프 이론의 일부로 19세기 후반에 발전하기 시작하였으며, 그래프에서 가장 중요한 클래스
 - 비선형 자료구조



트리

■ 수학에서 정의하는 트리

정의 8-1 트리

A 를 유한집합, T 를 A 위에서의 한 관계라고 하자. 이때 A 안에 적당한 정점 v_0 이 존재해서 v_0 에서 $A - \{v_0\}$ 의 모든 정점으로의 유일한 경로가 존재하고 v_0 에서 v_0 으로의 경로는 존재하지 않을 때, T 를 A 위에서의 **트리^{tree}** 혹은 **수형도**라고 한다.

정리 8-1 트리의 성질

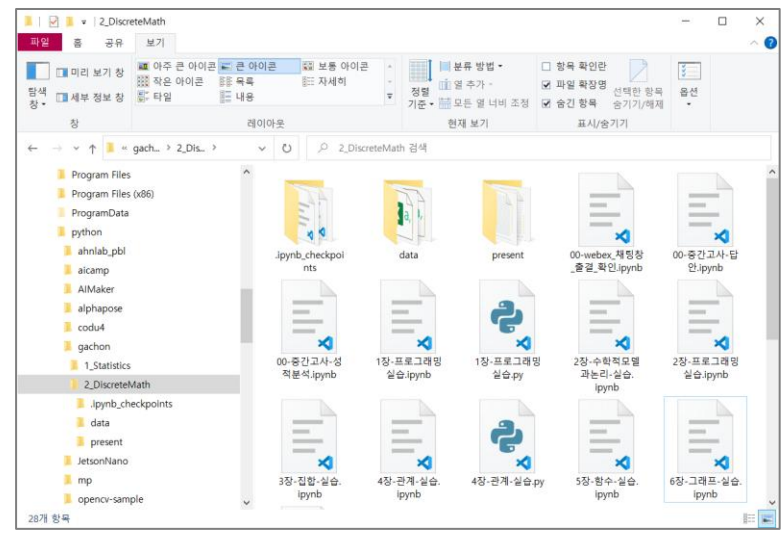
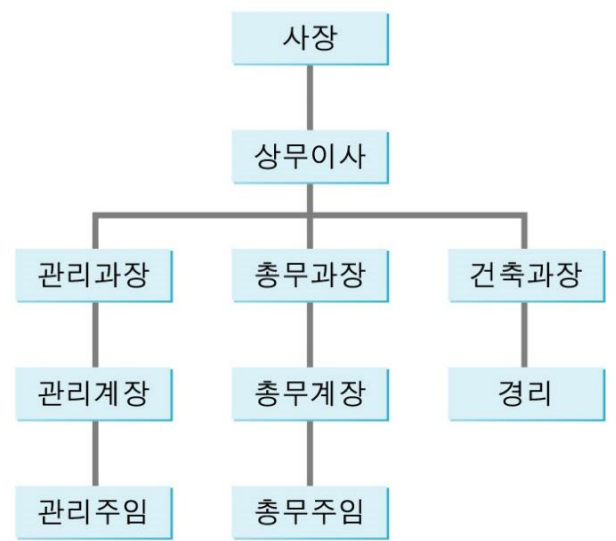
(T, v_0) 을 트리라 할 때 다음이 성립한다.

- (1) T 에는 어떤 순환도 존재하지 않는다.
- (2) v_0 은 유일한 루트 노드이다.
- (3) v_0 의 내차수는 0이고 나머지 정점들의 내차수는 1이다.

트리 구조

- 트리 구조로 표현하기 적합한 데이터의 특징

조건	설명
계층적 관계	상위/하위 개념이 뚜렷한 구조
부모 → 여러 자식 연결	단방향 관계, 순환 없음
검색 또는 분류가 중요한 데이터	빠른 탐색, 구조적 분할

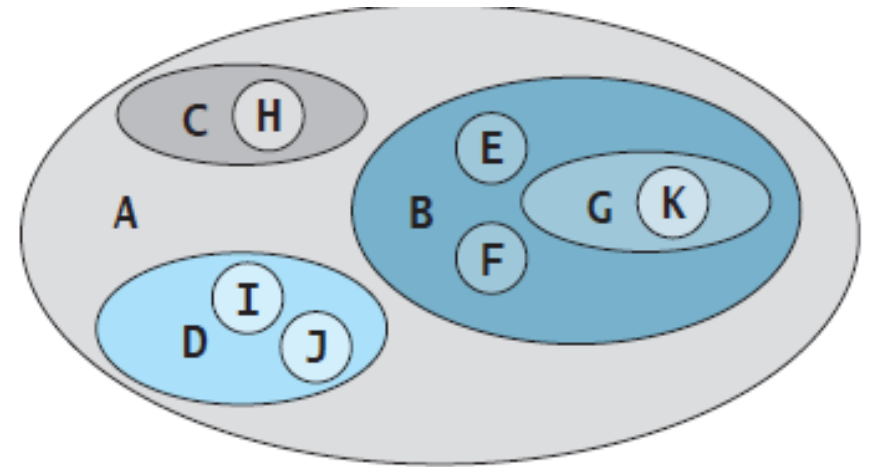
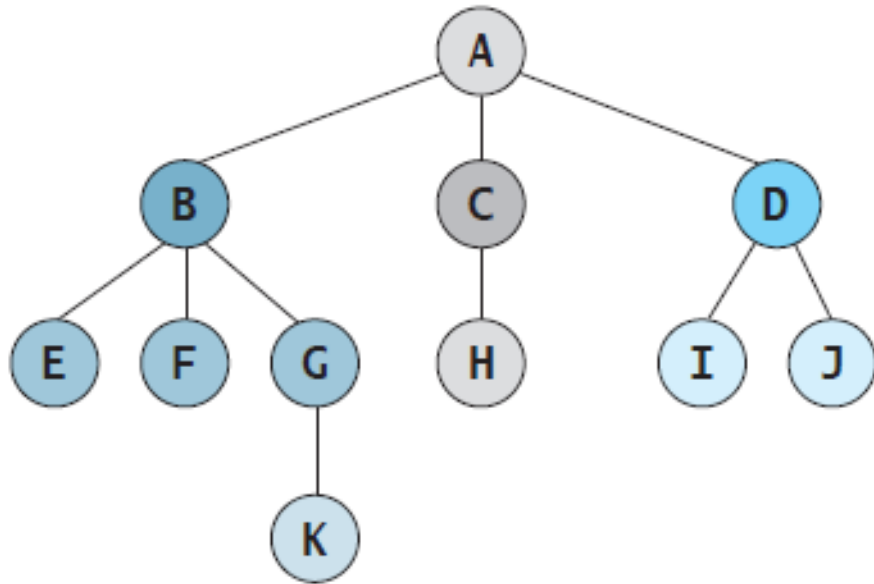


[Quiz]

- 실생활에서 트리 구조로 표현하기 좋은 데이터는?

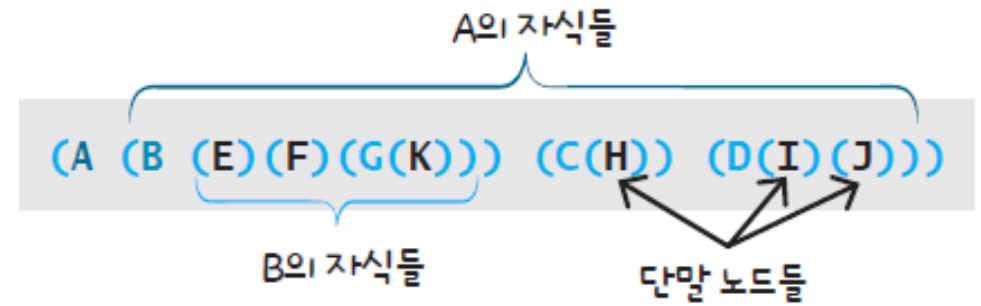
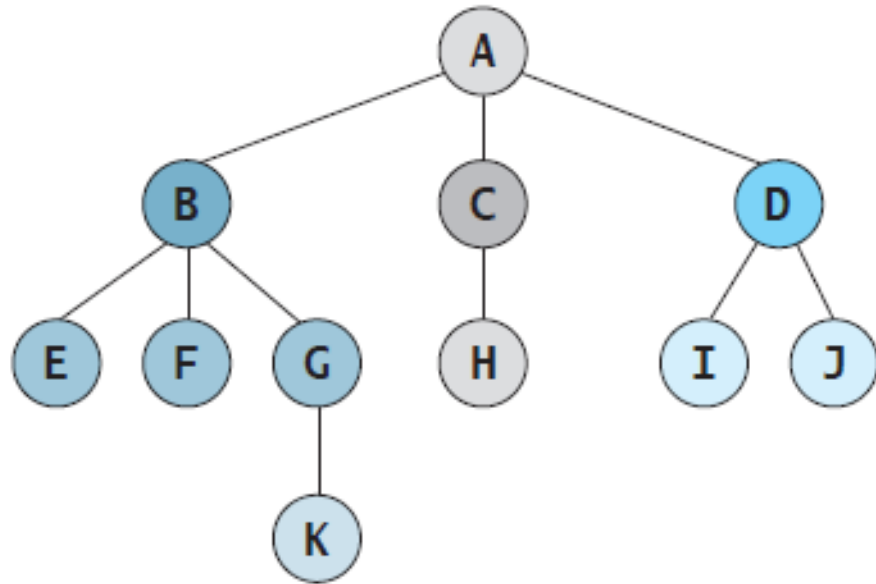
트리 구조의 표현 방법

- 노드와 간선의 연결 관계 : 중첩된 집합



트리 구조의 표현 방법

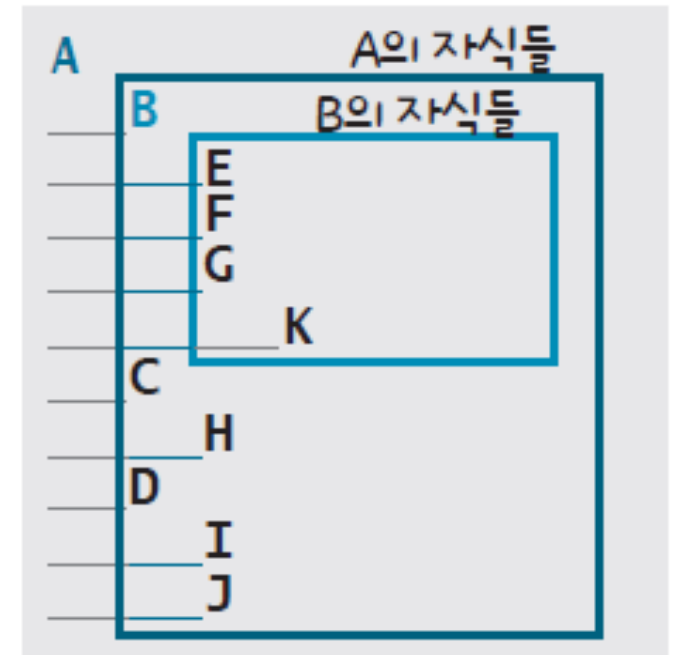
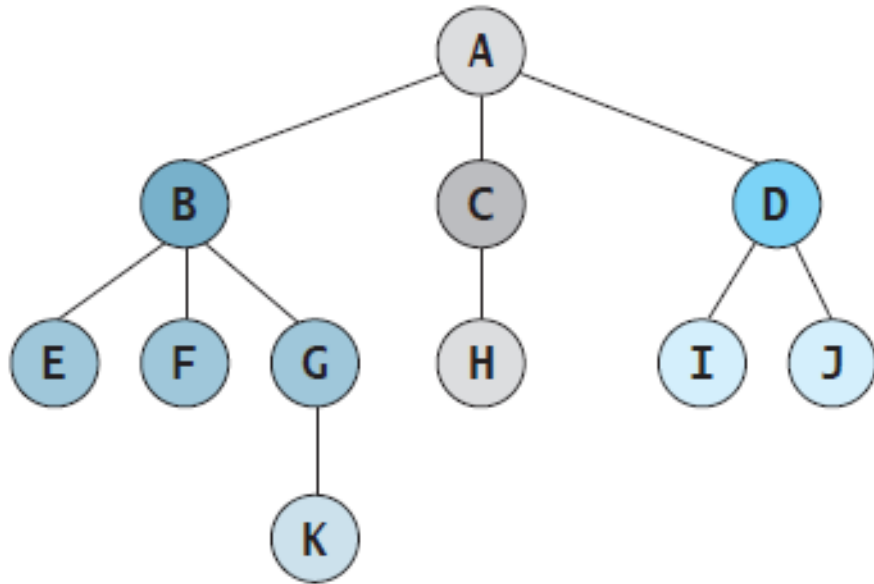
- 노드와 간선의 연결 관계 : 중첩된 괄호



→ 트리의 구조를 선형 문자열로 저장하거나 전송할 때 매우 유용한 표현

트리 구조의 표현 방법

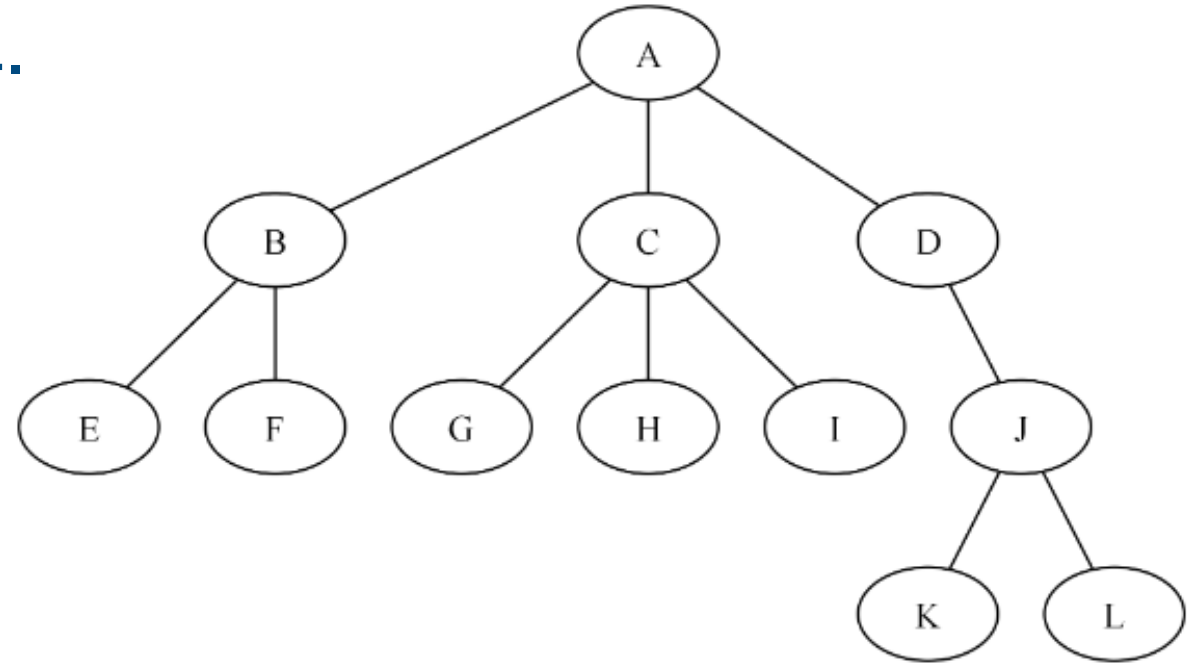
- 노드와 간선의 연결 관계 : 들여쓰기(indentation)



[Quiz]

- 트리를 다음 방법으로 표현해보세요.

- 1) 중첩된 집합으로 표현
- 2) 중첩된 괄호로 표현

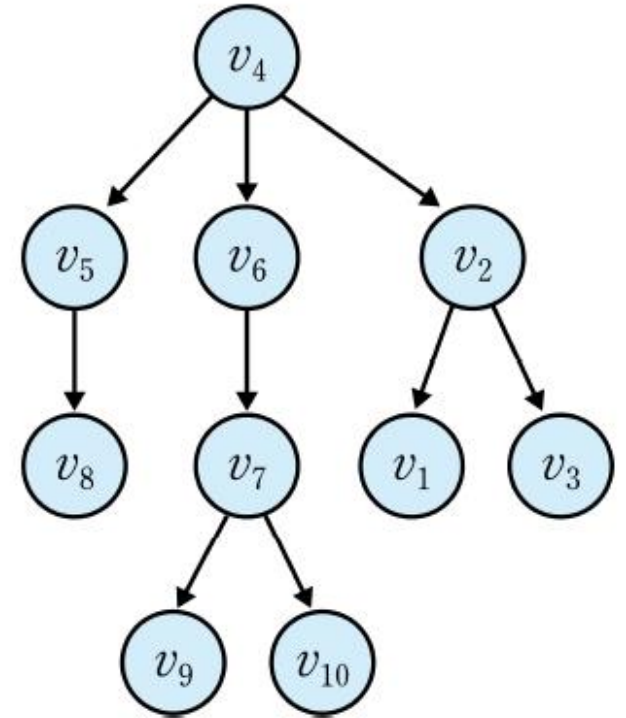


다양한 트리 구조

종류	특징	활용
일반 트리	각 노드가 제한 없이 여러 자식을 가질 수 있음.	계층적 데이터 표현.
이진 트리	각 노드가 최대 두 개의 자식 노드를 가짐.	기본적인 트리 구조.
포화 이진 트리	모든 노드가 0개 또는 2개의 자식을 가지며, 모든 레벨이 꽉 찬 상태.	학습 자료 모델링 등 간단한 구조.
완전 이진 트리	마지막 레벨을 제외하고 모든 레벨이 완전히 채워지고, 마지막 레벨은 왼쪽부터 순서대로 채워짐.	힙 자료구조 구현에 사용.
균형 이진 트리	왼쪽과 오른쪽 서브트리의 높이 차이가 1 이하로 유지됨.	검색 및 수정이 중요한 경우.
이진 탐색 트리 (BST)	왼쪽 서브트리는 루트보다 작은 값, 오른쪽 서브트리는 루트보다 큰 값을 가짐.	효율적인 검색, 삽입, 삭제 및 정렬.
AVL 트리	균형을 유지하며, 삽입/삭제 시 균형을 맞추기 위해 회전 연산을 수행함.	고성능 검색 및 메모리 관리.
레드-블랙 트리	삽입/삭제를 빠르게 처리하며 균형을 유지하는 이진 탐색 트리.	STL(Map, Set), 데이터베이스 구현.
트라이 (Trie)	문자열의 각 문자를 노드로 표현하여 빠른 탐색 가능.	자동 완성, 검색 추천, 사전 구현.
세그먼트 트리	특정 구간의 합, 최소값, 최대값 등을 빠르게 계산 가능.	구간 쿼리 및 업데이트에 사용.

트리의 기본 구조

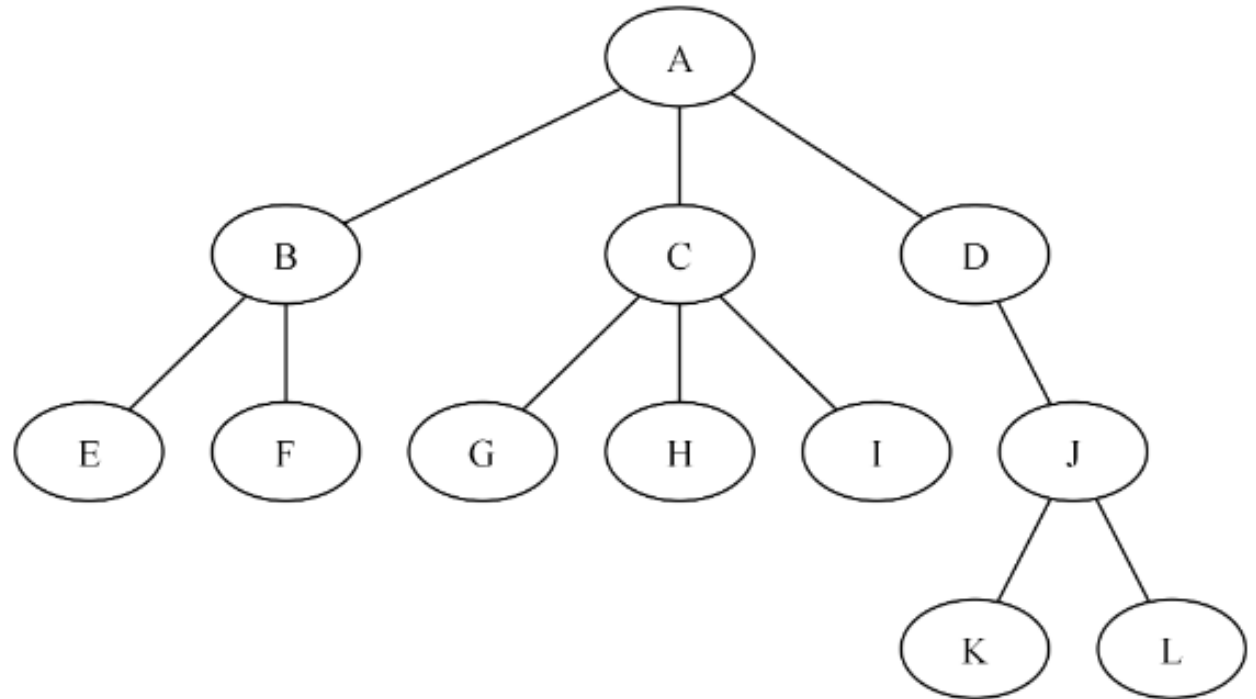
- 루트(Root): 트리의 최상위 노드
- 노드(Node): 데이터 단위, 자식 노드를 가질 수 있음
- 간선(Edge): 노드 간의 연결선
- 서브트리(Subtree): 어떤 노드를 루트로 하는 하위 트리
- 리프(Leaf): 자식이 없는 노드
- 높이(Height): 루트에서 가장 깊은 노드까지의 거리(간선수)
- 차수(Degree): 한 노드가 가진 자식 노드 수(가장 큰 차수)



[Quiz]

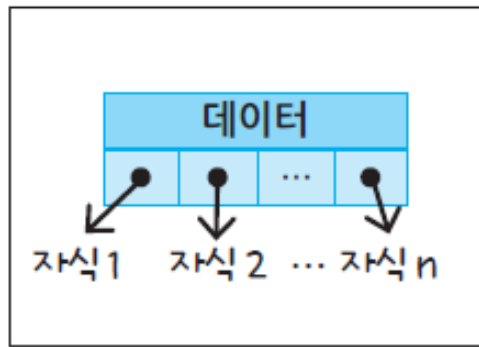
- 오른쪽 트리에서 다음을 구해보세요.

- 1) 루트 노드
- 2) 노드 J의 부모 노드
- 3) 노드 G의 형제 노드
- 4) 노드 C의 차수
- 5) 트리의 높이(간선기준)
- 6) 트리의 차수

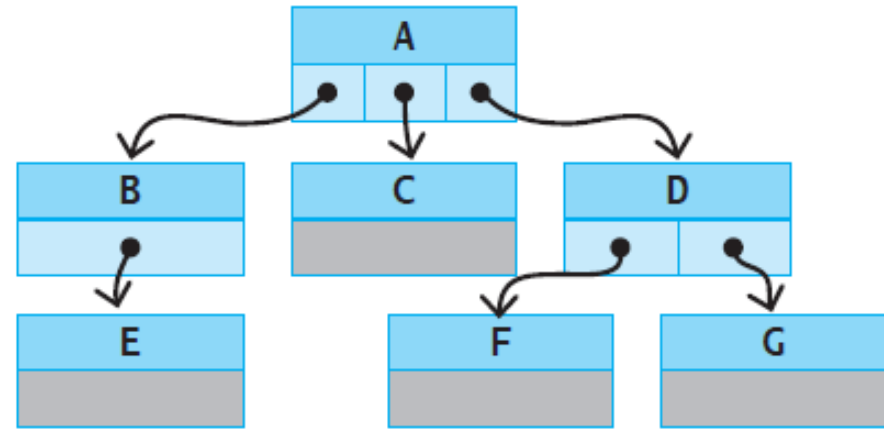
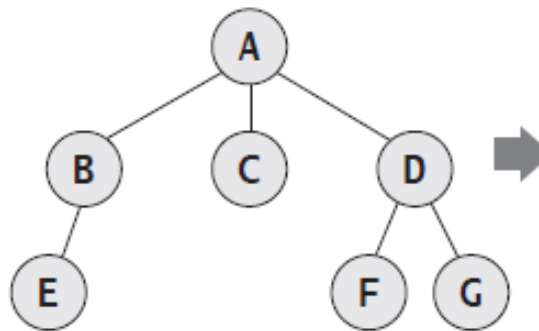


일반 트리의 표현

- 일반 트리(General tree) : 자식의 개수에 제한이 없는 트리
- 방법1: N-링크 표현



N-링크 노드의 구조



N-링크 표현의 예

일반 트리의 표현

N-링크 표현 트리 생성

```
class TreeNode:
    def __init__(self, data):
        self.data = data
        self.children = []

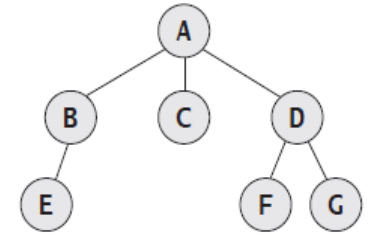
    def add_child(self, child_node):
        self.children.append(child_node)
```

노드 생성

```
A = TreeNode('A')
B = TreeNode('B')
C = TreeNode('C')
D = TreeNode('D')
E = TreeNode('E')
F = TreeNode('F')
G = TreeNode('G')
```

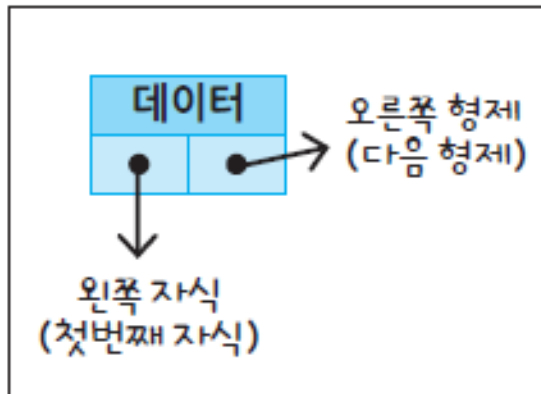
트리 연결 (N-링크 방식)

```
A.add_child(B)
A.add_child(C)
A.add_child(D)
B.add_child(E)
D.add_child(F)
D.add_child(G)
```

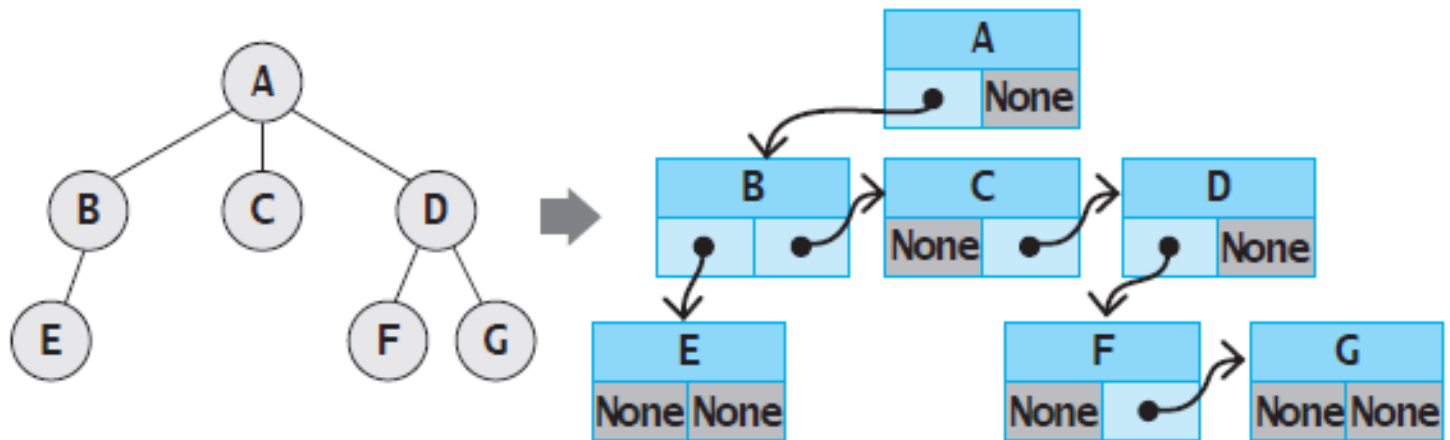


일반 트리의 표현

- 방법2 : 왼쪽 자식-오른쪽 형제 표현



왼쪽 자식-오른쪽 형제 표현의 예



왼쪽 자식-오른쪽 형제 표현의 예

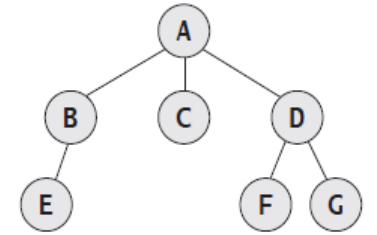
일반 트리의 표현

LCRS 표현 생성

```
class LCRSNode:
    def __init__(self, data):
        self.data = data
        self.left = None    # 왼쪽 자식
        self.right = None   # 오른쪽 형제
```

노드 생성

```
A = LCRSNode('A')
B = LCRSNode('B')
C = LCRSNode('C')
D = LCRSNode('D')
E = LCRSNode('E')
F = LCRSNode('F')
G = LCRSNode('G')
```



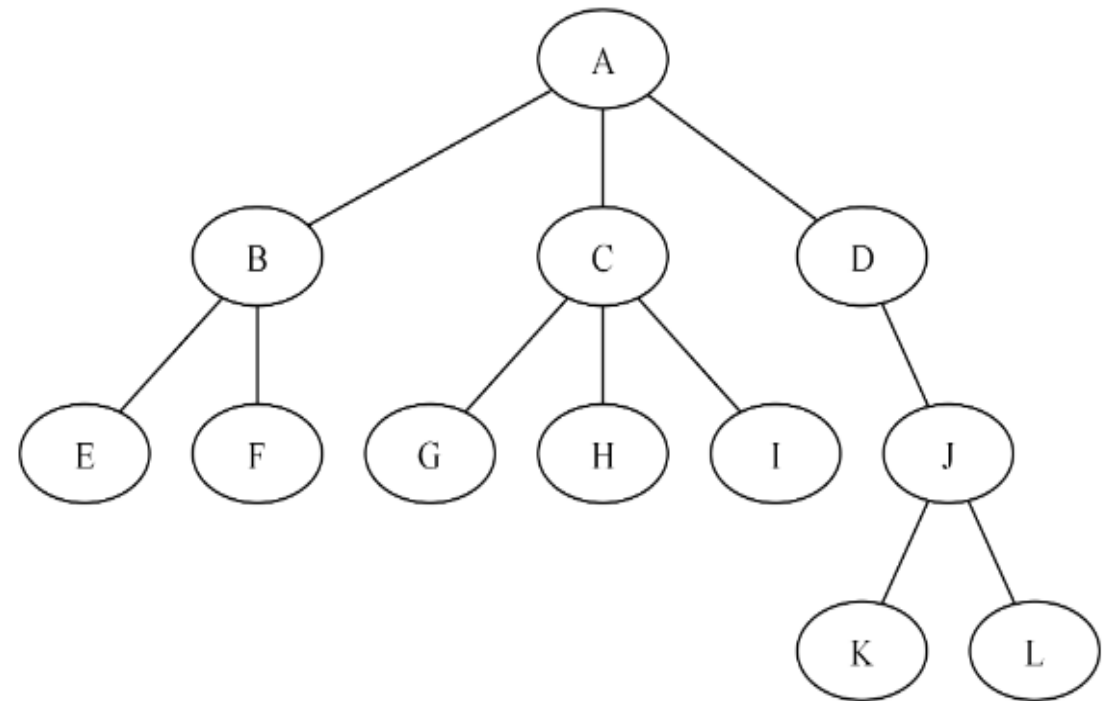
LCRS 연결

```
A.left = B      # A의 첫 자식 → B
B.right = C      # B의 형제 → C
C.right = D      # C의 형제 → D
B.left = E       # B의 첫 자식 → E
D.left = F       # D의 첫 자식 → F
F.right = G      # F의 형제 → G
```

실습 : 트리 구현하기

- 파이썬으로 트리 클래스를 구현하고 해당 값을 출력하세요.

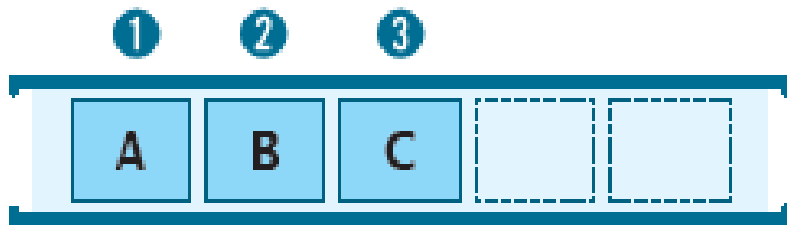
- 1) 전체 노드의 수
- 2) 트리의 높이(노드 기준)
- 3) 트리의 높이(간선 기준)
- 4) 루트 노드
- 5) 노드 J의 부모 노드
- 6) 노드 G의 형제 노드
- 7) 트리의 차수
- 8) 노드 C의 차수



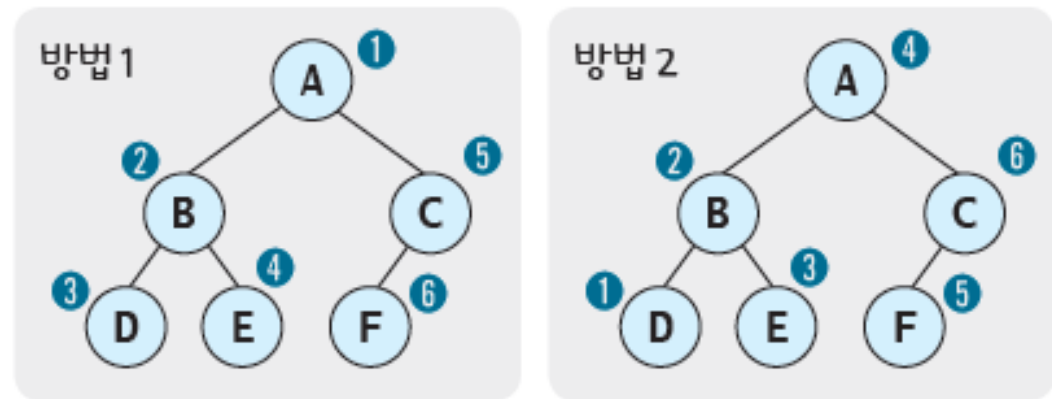
트리 기반 탐색

- 트리 기반 탐색(Tree-based Search)

- '트리'라는 계층적 자료구조를 활용하여 데이터를 탐색하는 알고리즘
- 노드와 간선으로 구성된 트리 구조를 순회하거나 특정 데이터를 찾기 위해 탐색하는 기법



선형자료구조는
순회 방법이 단순합니다.



트리는 다양한 방법으로 순회할 수 있습니다.

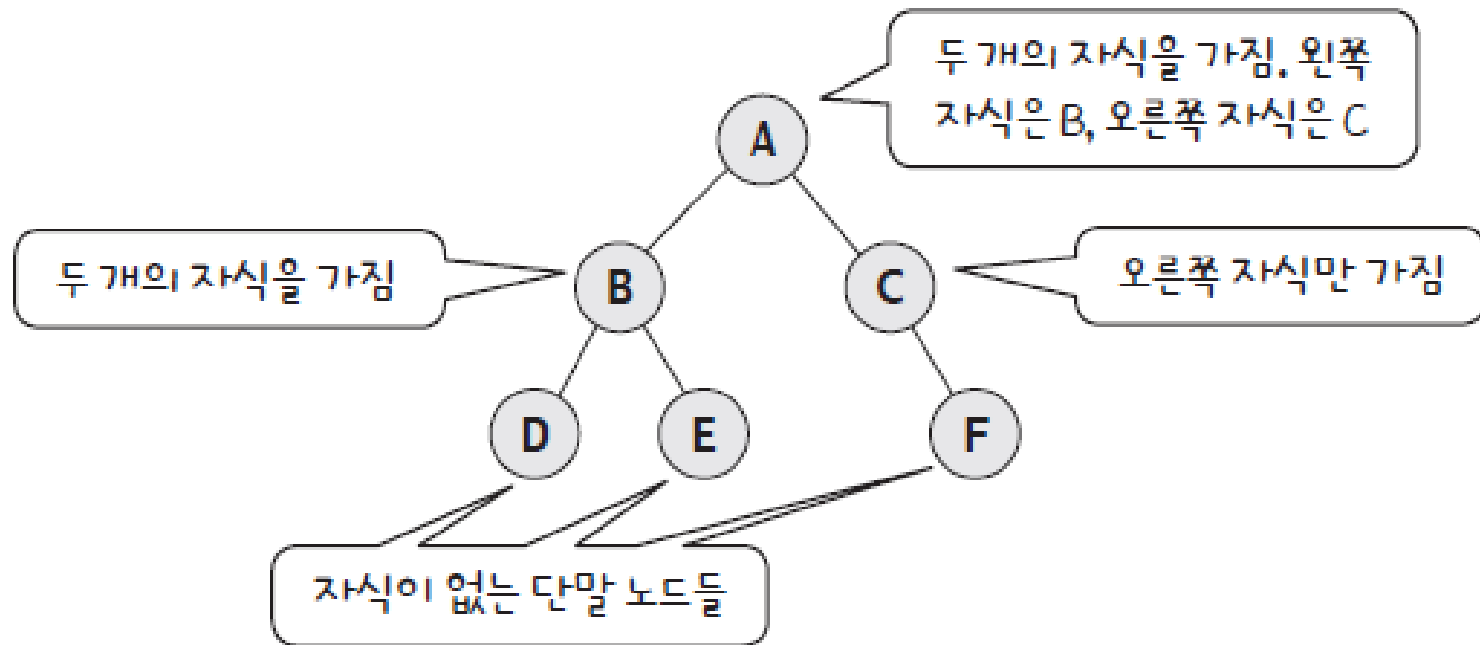
트리 기반 탐색 구조

구분	트리 순회(Tree Traversal)	이진 탐색 트리(Binary Search Tree)
목적	트리의 모든 노드를 특정 순서로 방문.	특정 키나 값을 가진 노드를 찾기 위한 방법.
주요 방식	<ul style="list-style-type: none">- 깊이 우선 탐색(DFS): 전위, 중위, 후위 순회- 너비 우선 탐색(BFS): 레벨 순회	<ul style="list-style-type: none">- 이진 탐색 트리(BST) 기반 탐색- B-트리, 트라이 등 다양한 탐색 알고리즘.
특징	<ul style="list-style-type: none">- 모든 노드를 방문하며 데이터를 정렬하거나 처리.- DFS는 스택(재귀) 사용, BFS는 큐 사용.	<ul style="list-style-type: none">- 특정 키를 빠르게 검색.- 탐색 트리는 균형 유지가 중요하여 효율적인 삽입/삭제 가능.
시간 복잡도	모든 노드를 한 번씩 방문: $O(n)$	균형 트리의 경우: $O(\log n)$ 비균형 트리는 최악의 경우: $O(n)$
활용 사례	<ul style="list-style-type: none">- 표현식 변환- 구조 분석	<ul style="list-style-type: none">- 데이터베이스 검색- 검색 시스템, 정렬 트리- 경로 찾기 및 네트워크 라우팅

이진 트리 (Binary Tree)

이진 트리

- 모든 노드가 최대 2개의 자식만을 가질 수 있는 트리
 - 모든 노드의 차수가 2 이하로 제한
 - 왼쪽 자식과 오른쪽 자식은 반드시 구별되어야 함



이진 트리

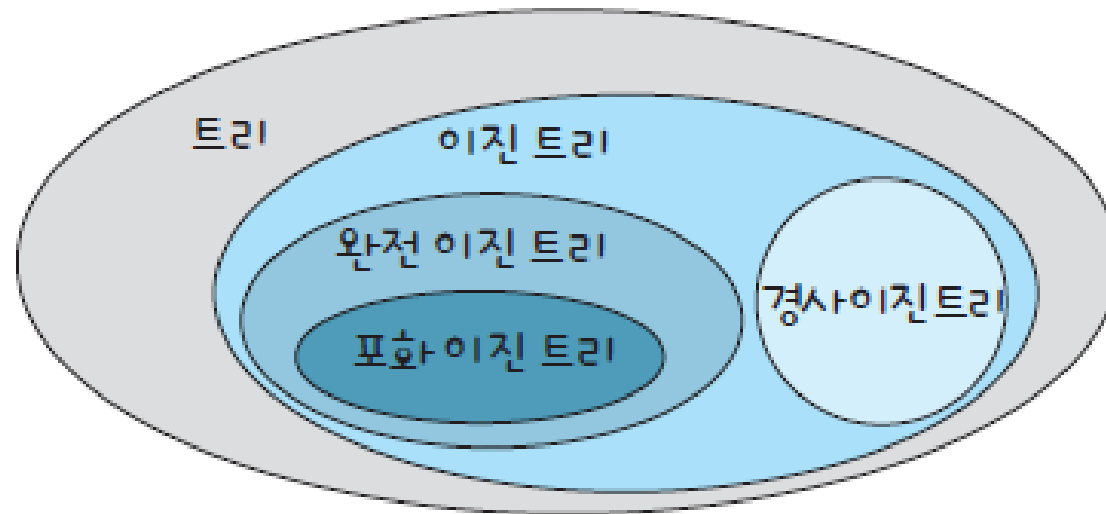
■ 이진 트리의 예

- 빠른 자료의 탐색이 가능한 이진 탐색 트리(Binary Search Tree)
- 수식을 트리 형태로 표현하여 계산하는 수식 트리 등
- 우선순위 큐를 효과적으로 구현하는 힙 트리(heap tree)

이진 트리의 종류

- 포화 이진 트리(full binary tree)
- 완전 이진 트리(complete binary tree)
- 균형 이진 트리(balanced binary tree)

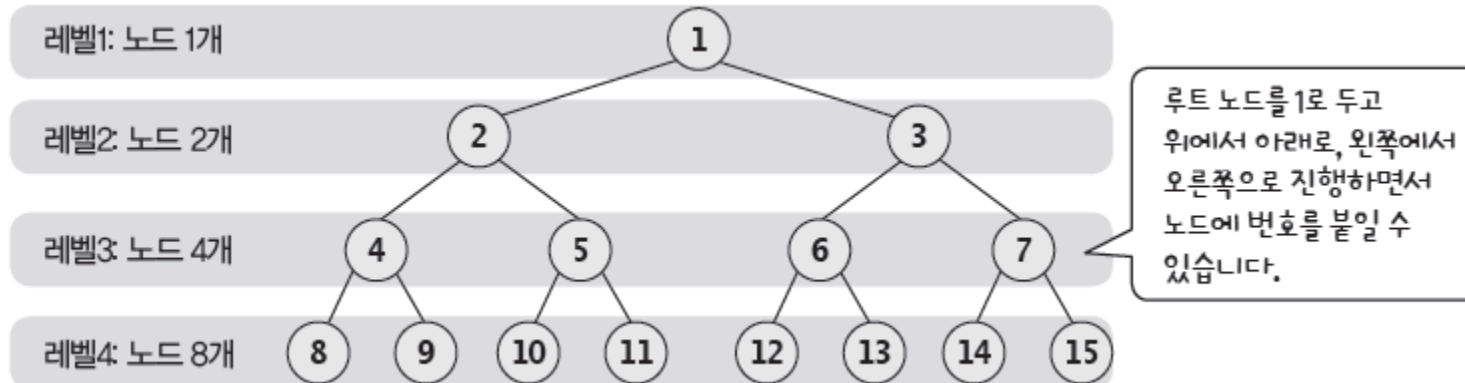
레벨 & 노드 수 관계에 따라 정의



이진 트리의 종류

■ 포화 이진 트리(full binary tree)

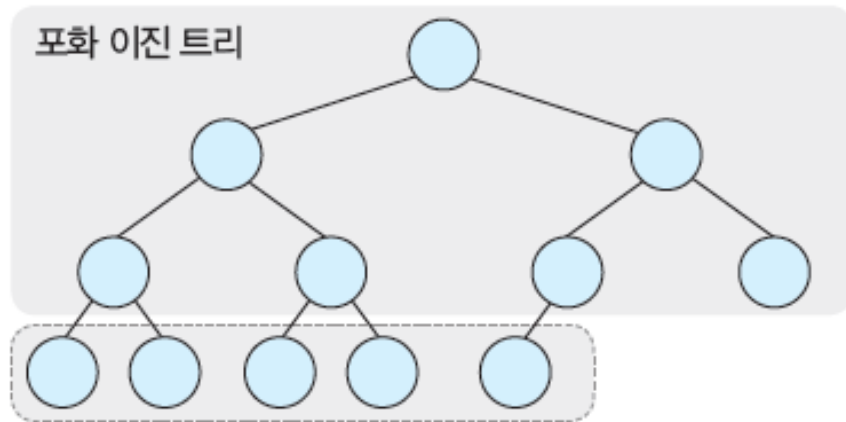
- 트리의 각 레벨에 **노드가 꽉 차 있는** 이진 트리
- 트리 높이(k)를 알면 전체 노드의 수를 쉽게 계산할 수 있음
- 각 노드에 순서대로 번호를 붙일 수 있음



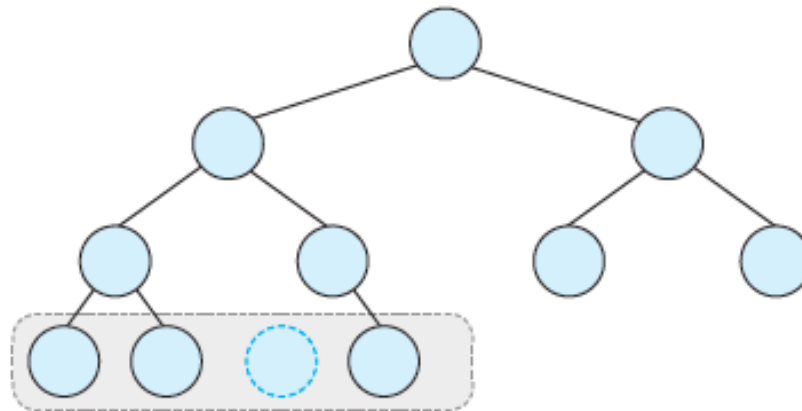
$$\text{전체 노드 개수} : 2^{1-1} + 2^{2-1} + 2^{3-1} + \dots + 2^{k-1} = \sum_{i=0}^{k-1} 2^i = 2^k - 1$$

이진 트리의 종류

- 완전 이진 트리(complete binary tree)
 - 레벨 $k-1$ 까지는 포화이진트리,
 - 마지막 레벨 k 에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워진 트리
 - heap은 완전 이진 트리의 대표적인 예



마지막 레벨이 순서대로 차 있음
→ 완전 이진 트리

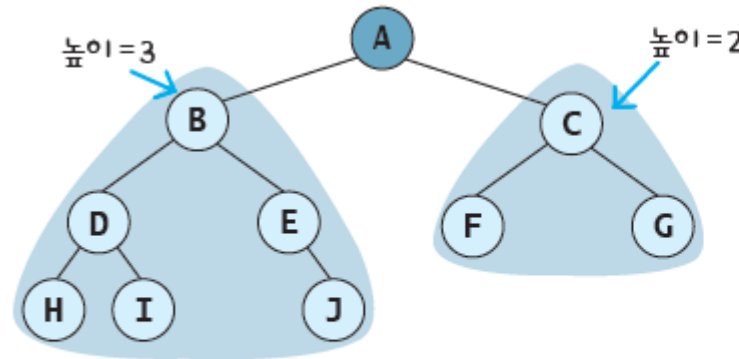


마지막 레벨에 빈 곳이 있음
→ 완전 이진 트리가 아님

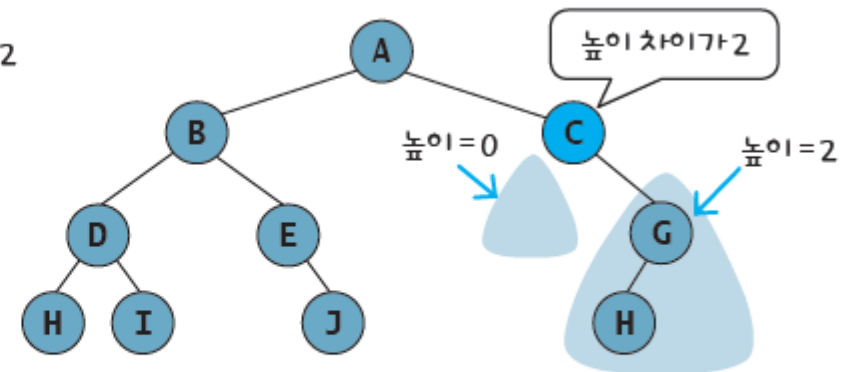
이진 트리의 종류

- 균형 이진 트리(balanced binary tree)

- '균형': 트리에서 좌우 서브 트리를 구분하기 때문에 균형 개념 적용
- 높이균형(or 균형) 이진 트리(height-balanced binary tree)는 모든 노드에서 좌우 서브 트리의 **높이 차이가 1 이하인 트리**
- 아니면 → 경사 트리



(a) 모든 노드의 좌우 서브 트리 높이 차이가 1 이하임 → **균형 이진 트리**



(b) **노드 C**의 좌우 서브 트리 높이 차이가 2임(1 초과) → **균형 이진 트리가 아님**

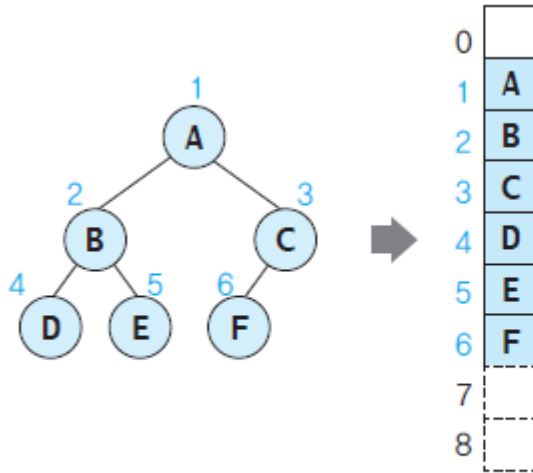
이진 트리의 표현 방법

- 배열 구조 표현(array data structure)
- 연결된 구조 표현(linked data structure)

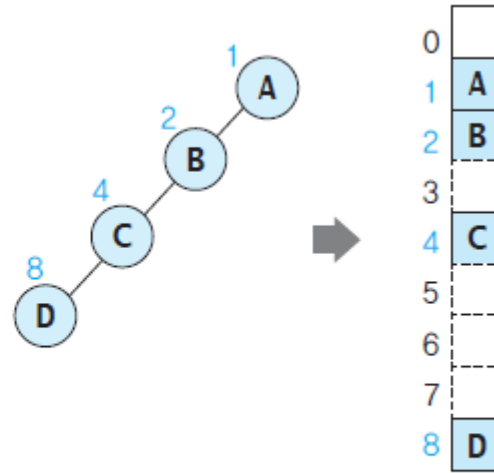
이진 트리의 표현

■ 배열 구조 표현

- 이진 트리를 **포화 이진 트리**의 일부라고 생각하고 번호 부여



(a) 완전 이진 트리의 배열 표현. 중간에 빈 칸이 발생하지 않음.



(b) 경사 이진 트리의 배열 표현. 중간에 빈 칸이 많이 발생할 수 있음.

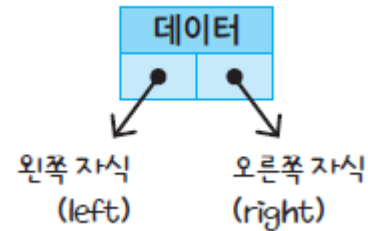
- 노드 i 의 부모 노드 인덱스 = $i/2$
- 노드 i 의 왼쪽 자식 노드 인덱스 = $2i$
- 노드 i 의 오른쪽 자식 노드 인덱스 = $2i+1$

파이썬에서는 나눗셈 연산자가 /와 //로 구분되어 있습니다. 정수 나눗셈을 위해서는 $i//2$ 를 써야합니다.

- ① 트리의 높이 구해 배열 할당
→ 길이가 $2^k - 1$ 인 배열
- ② 포화 이진 트리의 번호를 인덱스로 사용하여 배열에 노드 저장

이진 트리의 표현

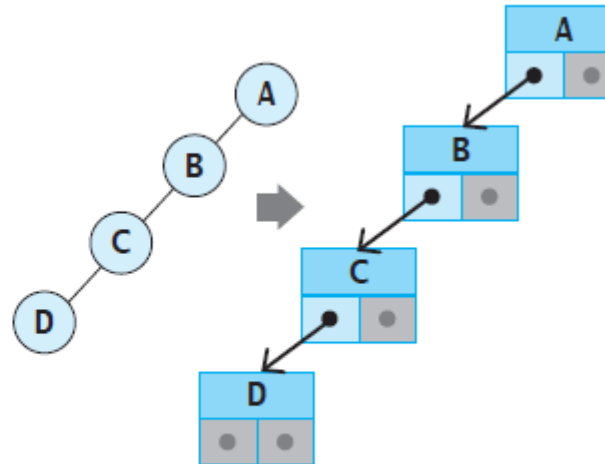
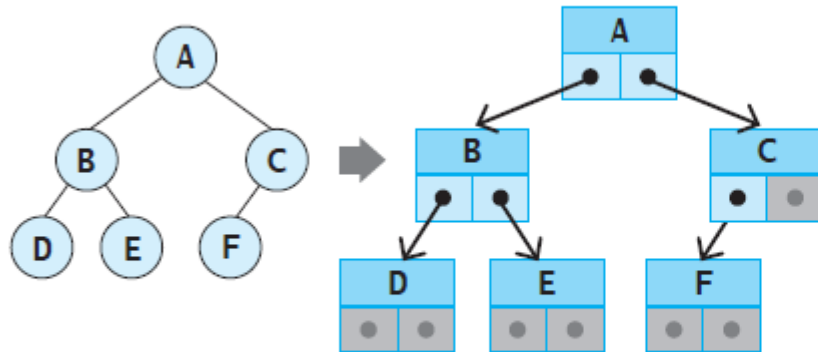
- 연결된 구조 표현: 링크 표현법
 - 연결된 구조의 이진 트리를 위한 노드의 구조



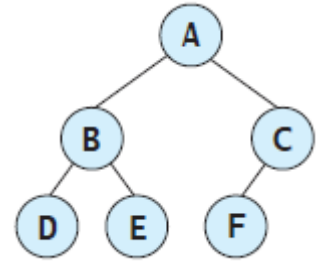
class BTreeNode:

```
def __init__(self, elem, left=None, right=None):  
    self.data = elem  
    self.left = left        # 왼쪽 자식을 위한 링크  
    self.right = right     # 오른쪽 자식을 위한 링크
```

이진 트리를 위한 노드의 생성자



이진 트리



이진 트리를 위한 노드 클래스

```
class BinaryTreeNode:
```

```
    def __init__(self, data, left=None, right=None):
```

```
        self.data = data        # 노드
```

```
        self.left = left        # 왼쪽 자식을 위한 링크
```

```
        self.right = right      # 오른쪽 자식을 위한 링크
```

```
    def isLeaf(self):
```

```
        return self.left is None and self.right is None
```

노드 생성

```
A = BinaryTreeNode("A")
```

```
B = BinaryTreeNode("B")
```

```
C = BinaryTreeNode("C")
```

```
D = BinaryTreeNode("D")
```

```
E = BinaryTreeNode("E")
```

```
F = BinaryTreeNode("F")
```

트리 연결 (A.left = B

```
A.right = C
```

```
B.left = D
```

```
B.right = E
```

```
C.left = F
```

[Quiz]

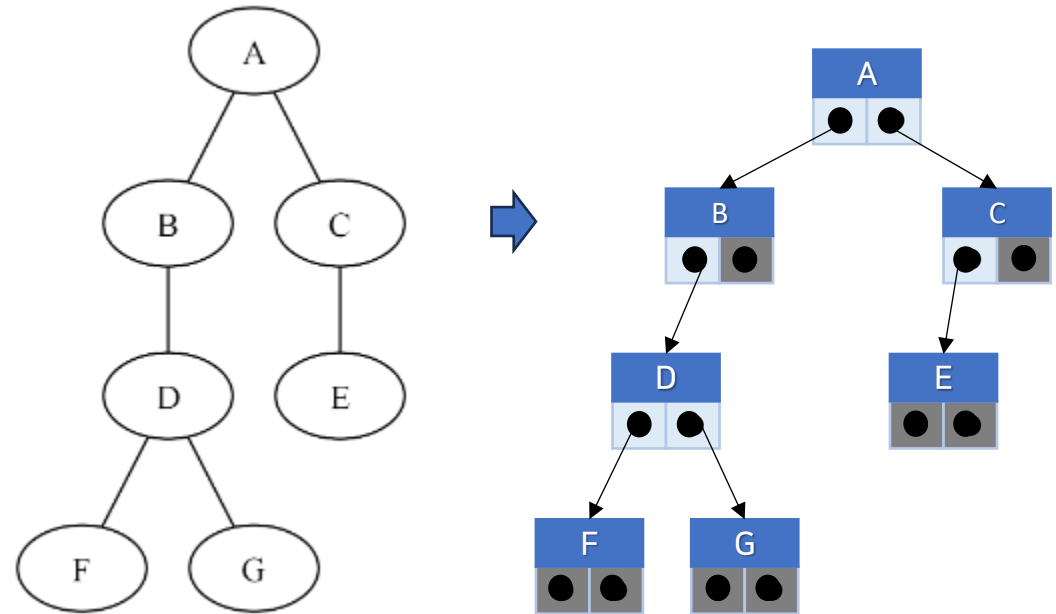
- 트리에 대해 답해보세요.

- 1) 이진 트리에서 노드의 수가 10개라면 간선의 수는?
- 2) 높이가 5인 포화 이진 트리의 노드 수는?
- 3) 높이가 5인 이진 트리의 최소 노드 수와 최대 노드 수는?
- 4) 링크 표현법으로 이진 트리를 표현할 때 노드의 개수가 n 이라면
최대의 None값 링크를 갖는 트리 이름과 그 때의 None값 링크 수는?

실습 : 이진 트리 구현하기

- 파이썬으로 이진 트리 클래스를 구현하고 해당 값을 출력하세요.

- 1) 전체 노드의 수
- 2) 트리의 높이(노드 기준)
- 3) 트리의 높이(간선 기준)
- 4) 루트 노드
- 5) 노드 D의 부모 노드
- 6) 노드 D의 형제 노드
- 7) 트리의 차수
- 8) 노드 C의 차수



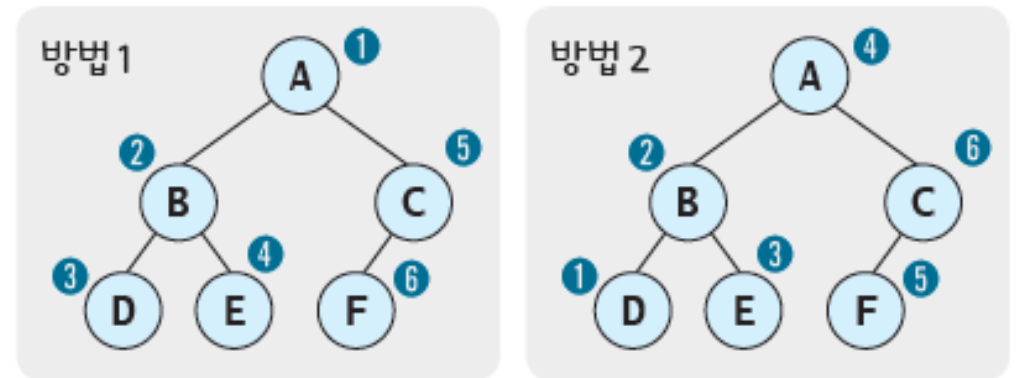
트리 순회

(Tree Traversal)

트리 순회

- 트리 순회(Tree Traversal)

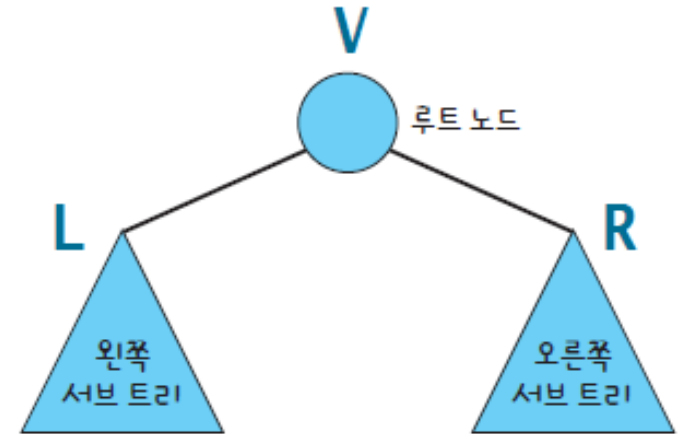
- 순회(탐방, Traversal)란 자료구조(트리, 그래프 등)의 모든 노드를 일정한 규칙에 따라 한 번씩 방문하는 알고리즘 또는 절차
- 전체 트리 구조를 순회하면서 모든 노드를 방문함
- ex) 전위, 중위, 후위, 레벨 순회



트리는 다양한 방법으로 순회할 수 있습니다.

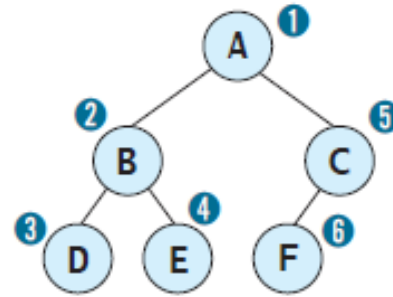
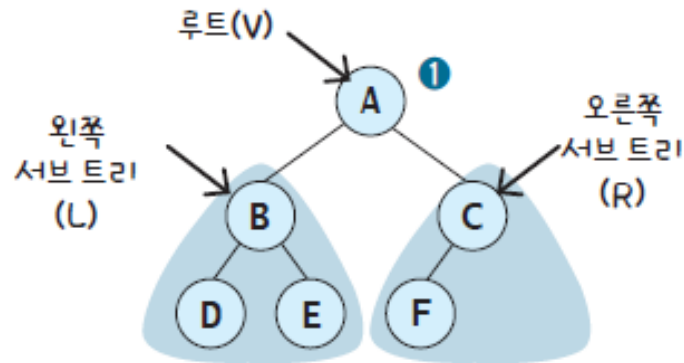
트리 순회

- 깊이 우선 탐색(DFS, Depth-First Search)
 - 노드를 깊게 파고들며 탐색함.
 - 재귀나 스택 사용
 - 전위 순회 (Preorder) : Root → Left → Right (VLR)
 - 중위 순회 (Inorder) : Left → Root → Right (LVR)
 - 후위 순회 (Postorder): Left → Right → Root (LRV)



전위 순회(preorder)

- $V \rightarrow L \rightarrow R$ 순서로 순회 진행 (서브 트리도 같은 순회 방법 적용)



```
def preorder(n) :
```

```
    if n is not None :
```

```
        print(n.data, end=' ')
```

```
        preorder(n.left)
```

```
        preorder(n.right)
```

전위순회 함수

노드를 방문해 처리할 연산들의 위치.

← 여기서는 노드의 데이터를 단순히 화면에 출력.

왼쪽 서브 트리 처리

오른쪽 서브 트리 처리

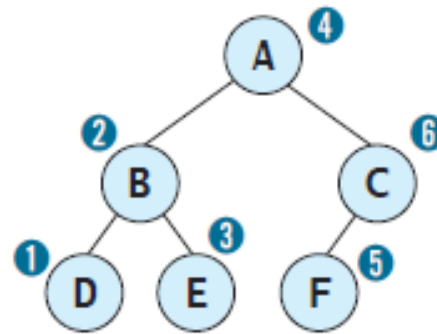
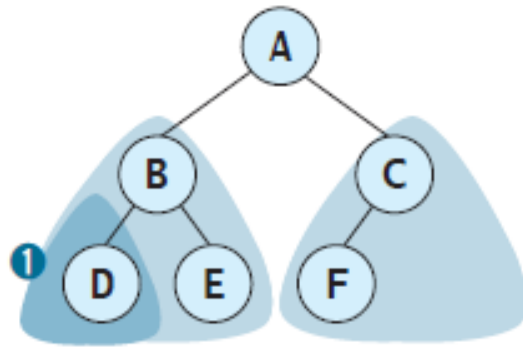
전위순회 결과 --> A B D E C F

중첩된 괄호표현 --> (A (B (D) (E)) (C (F)))

※본 강의 자료는 저작권이 있는 자료입니다. [그림 출처]: 자료구조와 알고리즘 with 파이썬 by 생능북스

중위 순회(inorder)

- $L \rightarrow V \rightarrow R$ 순서로 순회 진행



```
def inorder(n) :  
    if n is not None :  
        inorder(n.left)  
        print(n.data, end=' ' )  
        inorder(n.right)
```

전위순회 함수

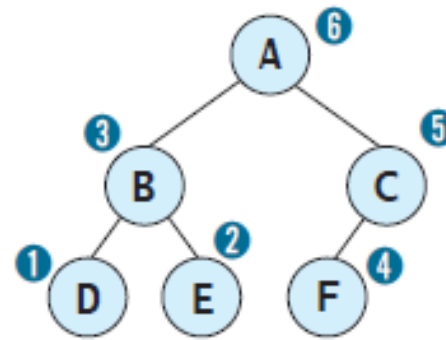
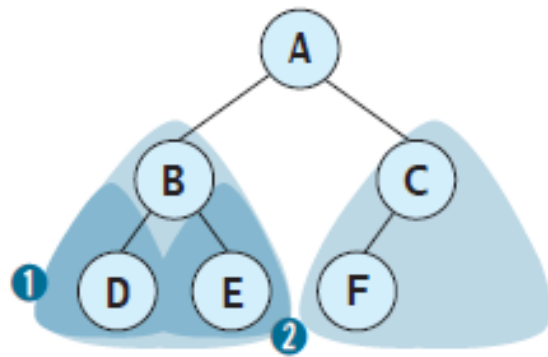
왼쪽 서브 트리 처리

← 노드에서 처리할 연산들의 위치

오른쪽 서브 트리 처리

후위 순회(postorder)

- $L \rightarrow R \rightarrow V$ 순서로 순회 진행



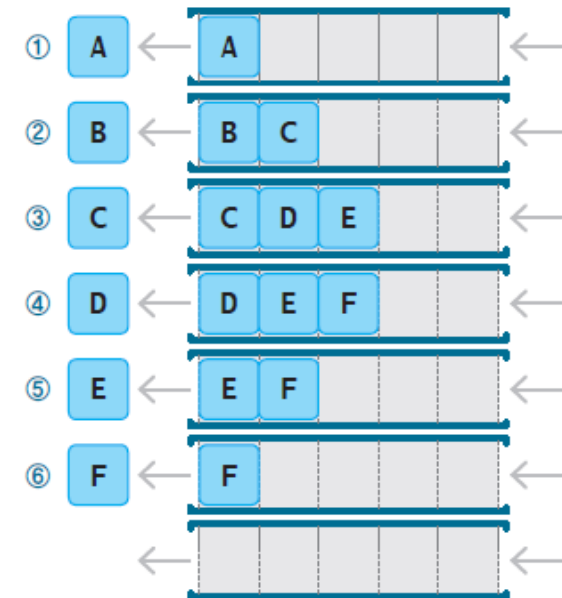
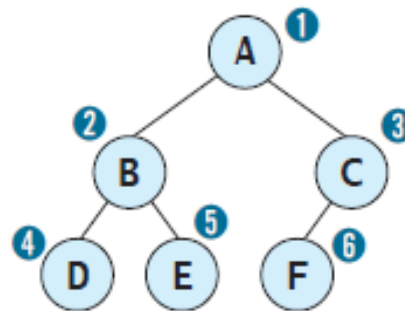
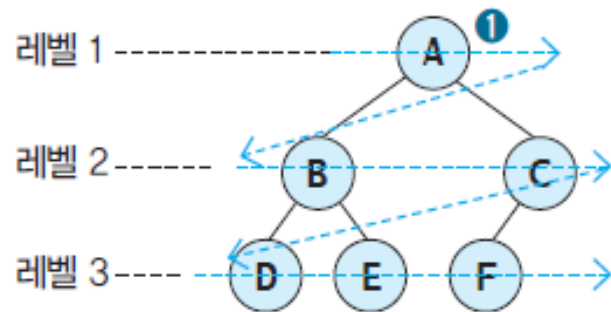
```
def postorder(n) :  
    if n is not None :  
        postorder(n.left)  
        postorder(n.right)  
        print(n.data, end=' ')
```

← 노드에서 처리할 연산들의 위치

트리 순회(Tree Traversal)

- 너비 우선 탐색(BFS, Breadth-First Search)

- 레벨 순서대로 위에서 아래, 왼쪽에서 오른쪽(루트에서 가까운 노드부터 넓게 탐색)
- 레벨 순회는 순환을 사용하지 않고 큐(Queue) 사용

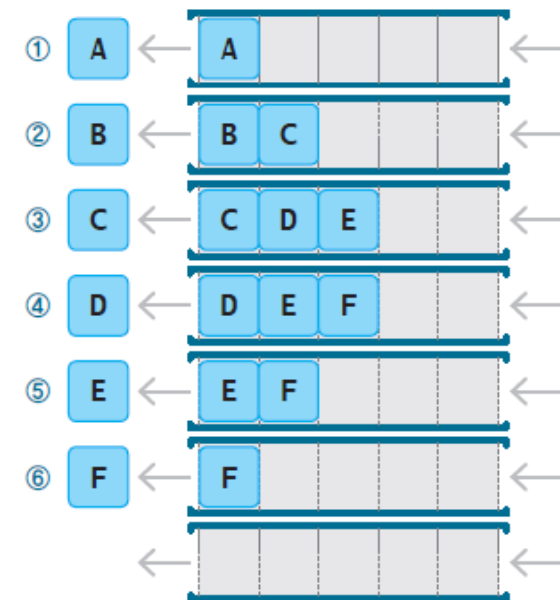


■ 너비 우선 탐색(BFS, Breadth-First Search)

이진트리의 레벨 순회

```
def levelorder(root) :  
    queue = ArrayQueue()  
    queue.enqueue(root)  
    while not queue.isEmpty() :  
        n = queue.dequeue()  
        if n is not None :  
            print(n.data, end=' ')  
            queue.enqueue(n.left)  
            queue.enqueue(n.right)
```

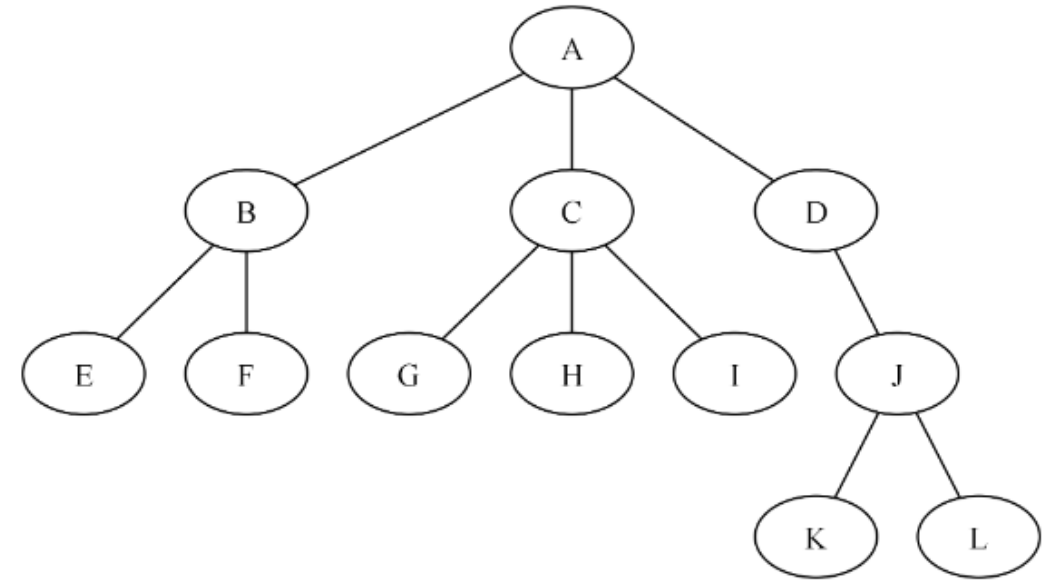
큐가 공백 상태가
될 때까지 반복



[Quiz]

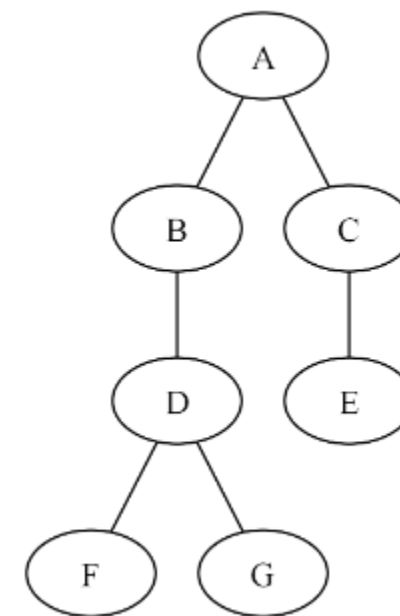
- 일반 트리에 적용이 어려운 트리 순회 방식은? 각각의 트리 순회 결과는?

- 1) 전위 순회 (Preorder)
- 2) 중위 순회 (Inorder)
- 3) 후위 순회 (Postorder)
- 4) 레벨 순회 (Level)



실습 : 이진 트리 순회하기

- 앞에서 만든 이진 트리 클래스에 기능을 추가하고 결과를 출력하세요.
 - 1) 전위 순회 결과를 출력하세요.
 - 2) 중위 순회 결과를 출력하세요
 - 3) 후위 순회 결과를 출력하세요
 - 4) 레벨 순회 결과를 출력하세요.



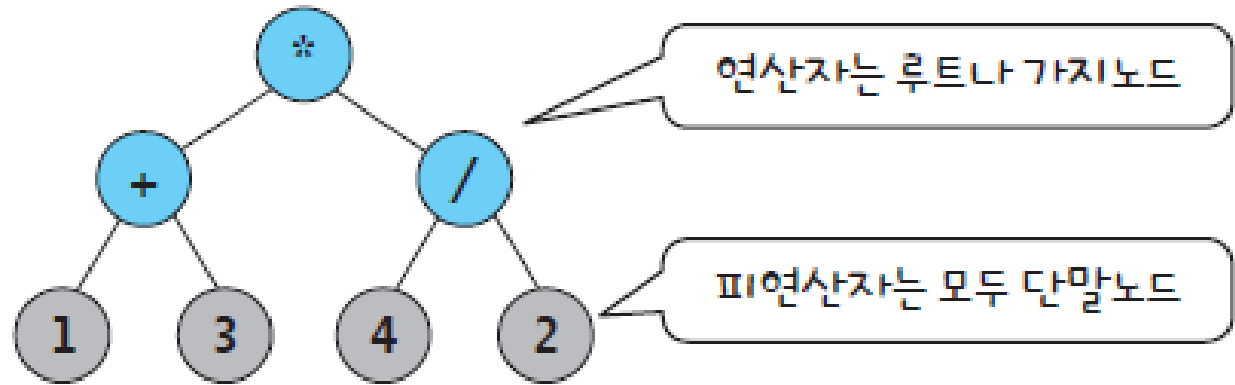
응용 : 수식 트리

수식 트리

- 수식 트리(Expression Tree)

- 산술식을 트리 형태로 표현한 이진 트리
- 수식 트리는 하나의 연산자가 두 개의 피연산자를 갖는다고 가정

$(1 + 3) * (4 / 2)$

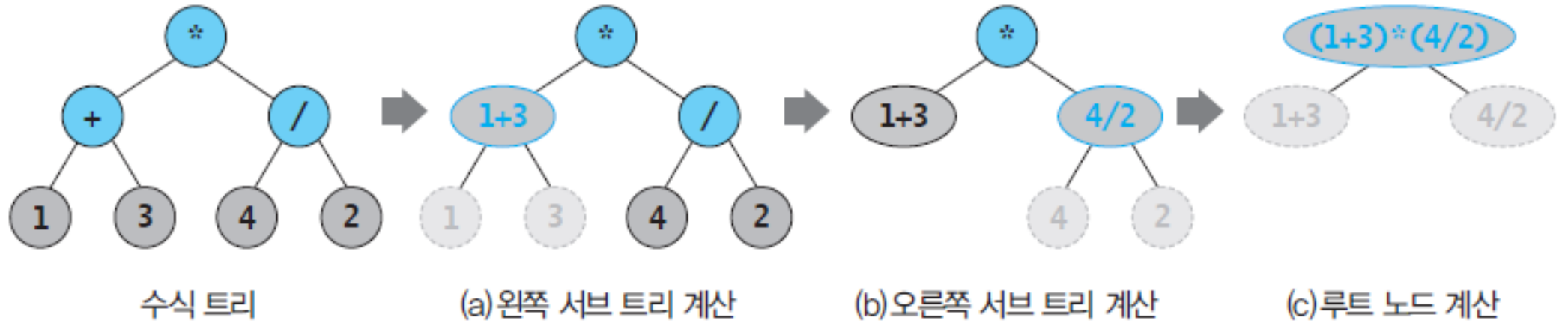


- 수식 트리의 계산
- 수식 트리 만들기

수식 트리의 계산

■ 수식 트리의 계산

- 어떤 연산자를 계산하려면 자식 노드의 계산이 반드시 끝나 있어야 한다.



- 수식 트리의 계산에는 후위 순회(postorder)가 사용된다.

수식 트리의 계산

- 수식 트리의 계산 (후위 순회)

수식트리 계산 함수

```
def evaluate(node) :
```

```
    if node is None :
```

```
        return 0
```

```
    elif node.isLeaf() :
```

```
        return node.data
```

```
    else :
```

```
        op1 = evaluate(node.left)
```

```
        op2 = evaluate(node.right)
```

```
        if node.data == '+' : return op1 + op2
```

```
        elif node.data == '-' : return op1 - op2
```

```
        elif node.data == '*' : return op1 * op2
```

```
        elif node.data == '/' : return op1 / op2
```

공백 트리면 0 반환

단말 노드이면 --> 피연산자

그 노드의 값(데이터) 반환

루트나 가지노드라면 --> 연산자

왼쪽 서브트리 먼저 계산

오른쪽 서브트리 먼저 계산

루트(현재 노드) 처리

수식의 표현 방법

■ 수식의 표현 방법

전위(prefix)	중위(infix)	후위(postfix)
연산자 피연산자1 피연산자2	피연산자1 연산자 피연산자2	피연산자1 피연산자2 연산자
+ A B	A + B	A B +
+ 5 * A B	5 + A * B	5 A B * +

사람이 수식 처리하는 방법

컴퓨터가 수식 처리하는 방법

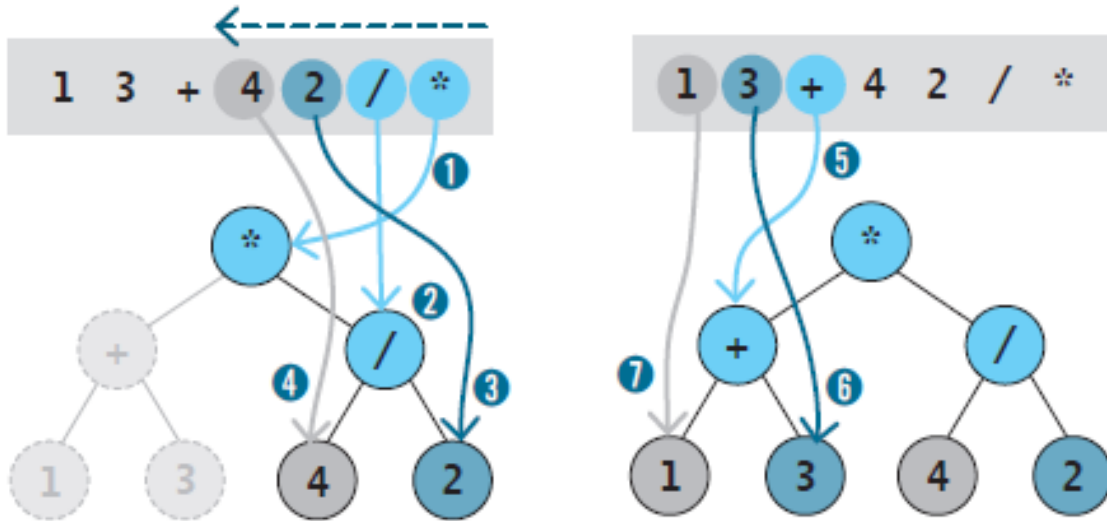
→ Shunting Yard Algorithm (by 다익스트라)

■ 후위 표기의 장점

- 괄호를 사용하지 않음 / 수식을 읽으면서 바로 계산
- 연산자의 우선순위를 생각할 필요가 없음

후위 표기 식으로 수식 트리 만들기

- 맨 뒤에서 앞으로 읽으면서 처리
- 입력 수식 : 1 3 + 4 2 / *

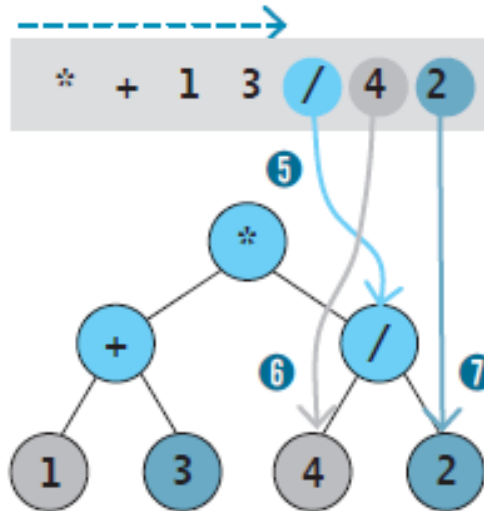
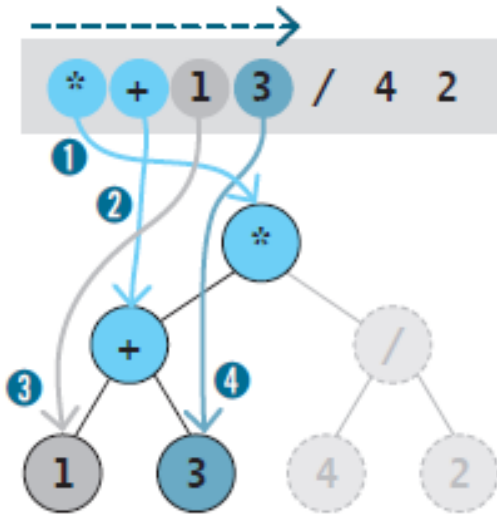


후위표기 수식을 이용한 수식트리 만들기

```
def buildETree( expr ):  
    if len(expr) == 0 :  
        return None  
  
    token = expr.pop()  
    if token in "+-*/" :  
        node = BTNode(token)  
        node.right = buildETree(expr)  
        node.left = buildETree(expr)  
        return node  
    else :  
        return BTNode(float(token))
```

전위 표기 식으로 수식 트리 만들기

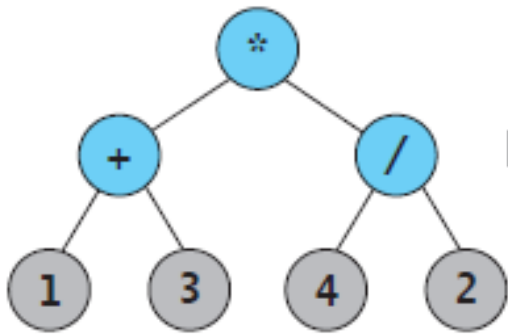
- 맨 앞에서 뒤로 읽으면서 처리
- 입력 수식 : * + 1 3 / 4 2



```
def buildETree2( expr ):  
    if len(expr) == 0 :  
        return None  
  
    token = expr.pop(0)  
    if token in "+-*/" :  
        node = BTreeNode(token)  
        node.left = buildETree2(expr)  
        node.right = buildETree2(expr)  
        return node  
    else :  
        return BTreeNode(float(token))
```

실습 : 수식 트리 테스트

- 앞에서 구현한 수식 트리를 파이썬으로 만들고 이진 트리 클래스를 이용하여 아래와 같이 트리의 연산 정보를 1.키보드로 후위표기식 입력을 받고 → 2.입력 받은 토큰을 분리하고→ 3.순회 정보와 함께 계산 결과를 출력하세요.



수식 트리

입력 (후위 표기): **1 3 + 4 2 / *** 키보드로 후위 표기식으로 수식 트리 입력

토큰분리(expr): ['1', '3', '+', '4', '2', '/', '*']

전위 순회: (* (+ (1.0) (3.0)) (/ (4.0) (2.0)))

중위 순회: 1.0 + 3.0 * 4.0 / 2.0

후위 순회: 1.0 3.0 + 4.0 2.0 / *

계산 결과 : 8.0

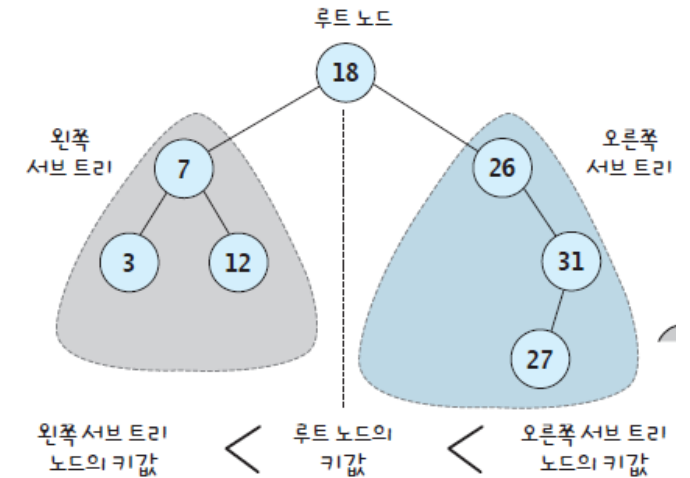
이진 탐색 트리 (Binary Search Tree, BST)

이진 탐색 트리

■ 이진 탐색 트리(Binary Search Tree: BST)

- BST는 "탐색(search)"에 특화된 이진 트리로, 각 노드가 다음 조건을 만족해야 함.

- ① 왼쪽 서브트리: 현재 노드보다 작은 값들
- ② 오른쪽 서브트리: 현재 노드보다 큰 값들
- ③ 이 구조는 모든 서브트리에도 동일하게 적용



- 이 구조 덕분에 탐색, 삽입, 삭제 연산이 빠름
- 중위 순회(Inorder)를 통해 트리의 모든 요소를 오름차순으로 출력할 수 있다.
- 최선/최악 시간 복잡도: $O(\log n)$ / $O(n)$

이진 탐색 트리 동작방식

■ 이진 탐색 트리 동작 방식 : 삽입(Insertion)

- 탐색에 실패한 위치에 새로운 노드를 삽입

삽입 연산 : Insertion

```
def insert(self, root, key):
```

```
    if root is None:
```

```
        return BSTNode(key)
```

```
    if key < root.key:
```

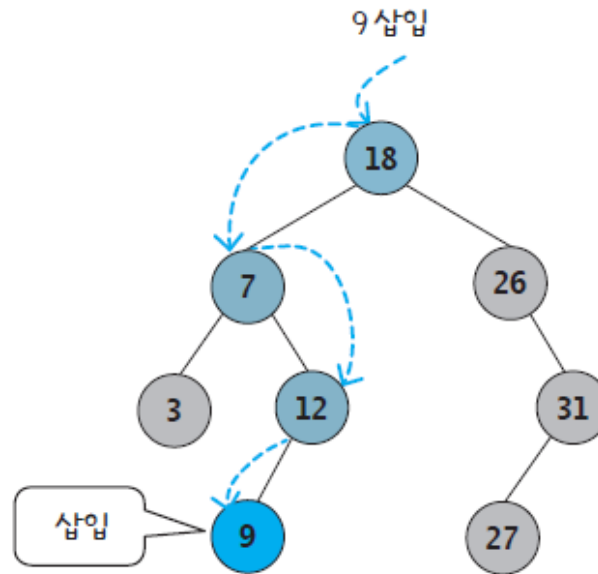
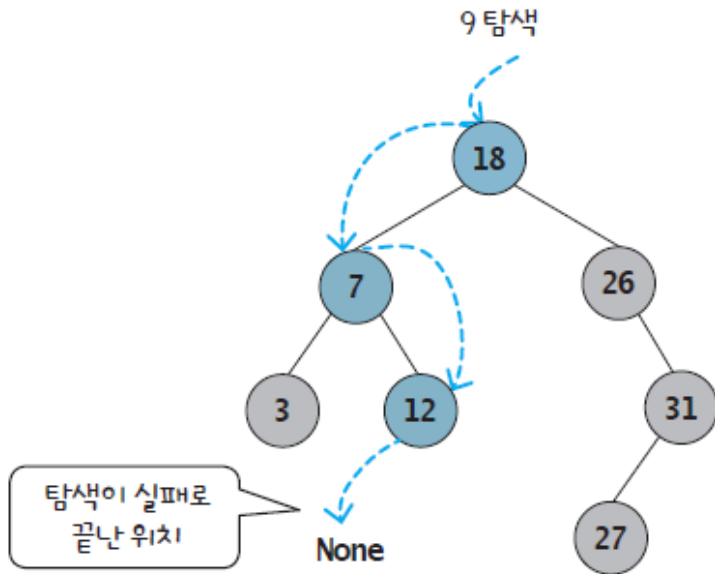
```
        root.left = self.insert(root.left, key)
```

```
    elif key > root.key:
```

```
        root.right = self.insert(root.right, key)
```

```
    # 중복은 삽입하지 않음
```

```
    return root
```



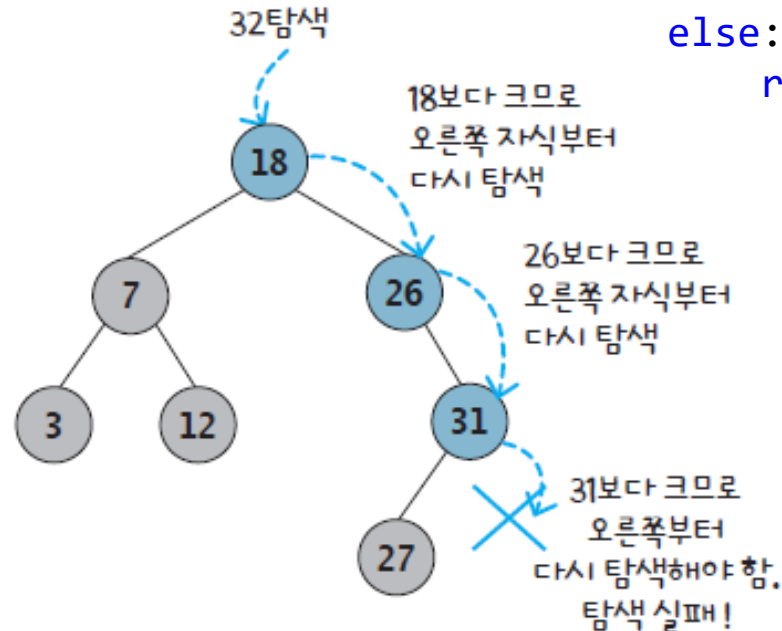
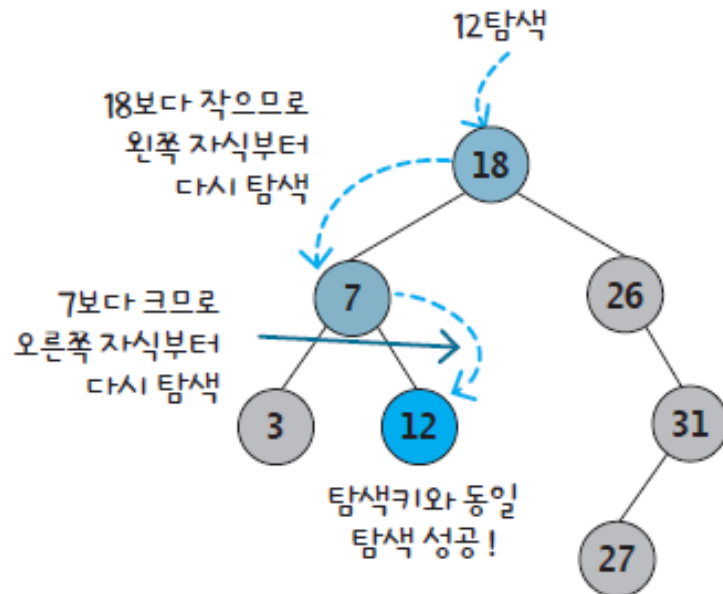
이진 탐색 트리 동작방식(연산)

■ 이진 탐색 트리 동작 방식 : 탐색(Search)

- 루트 노드에서 시작해서 아래로 내려감
- ex) 12의 탐색과 32의 탐색

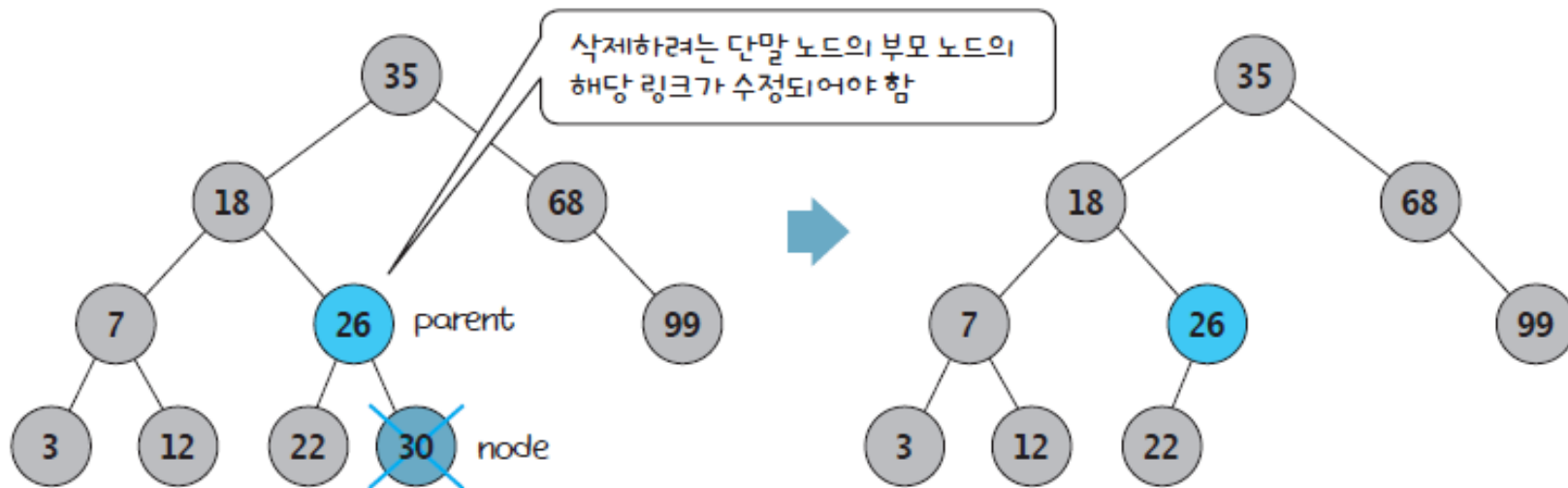
탐색 연산 : Search

```
def search(self, root, key):  
    if root is None or root.key == key:  
        return root  
    if key < root.key:  
        return self.search(root.left, key)  
    else:  
        return self.search(root.right, key)
```



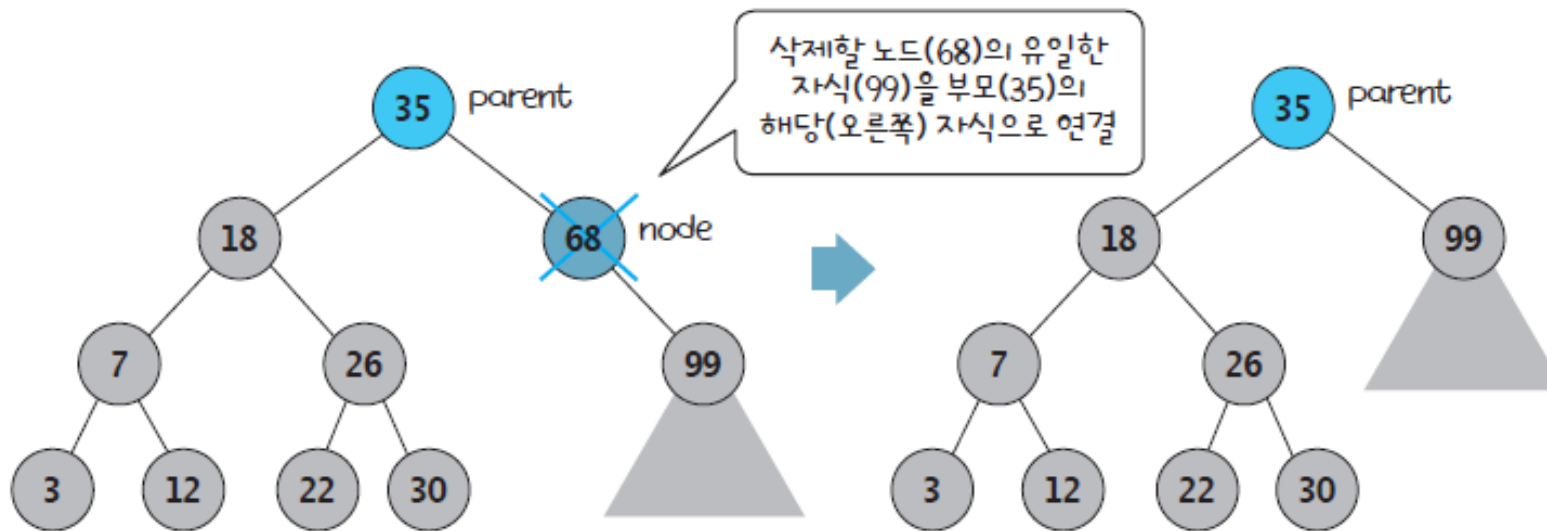
이진 탐색 트리 동작방식

- 이진 탐색 트리 동작 방식 : 삭제>Delete
- 삭제할 노드의 자식 수에 따라 3가지 경우로 구분
- Case 1: 자식이 없는 경우-단말 노드의 삭제 (노드 바로 삭제)



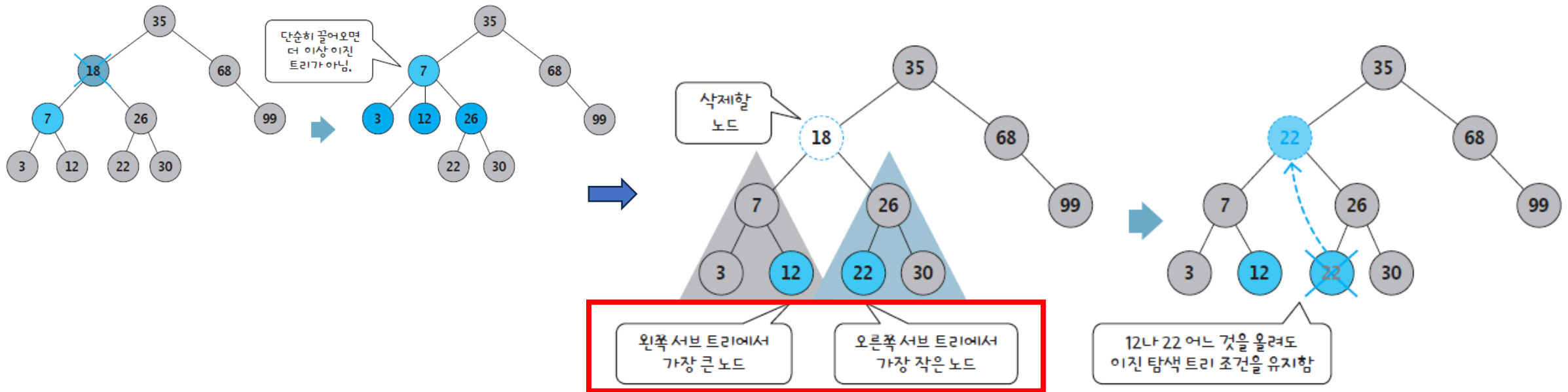
이진 탐색 트리 동작방식

- 이진 탐색 트리 동작 방식 : 삭제>Delete)
 - Case 2: 자식이 하나인 노드의 삭제(노드 삭제 후 자식 노드를 삭제 노드 위치로 이동)



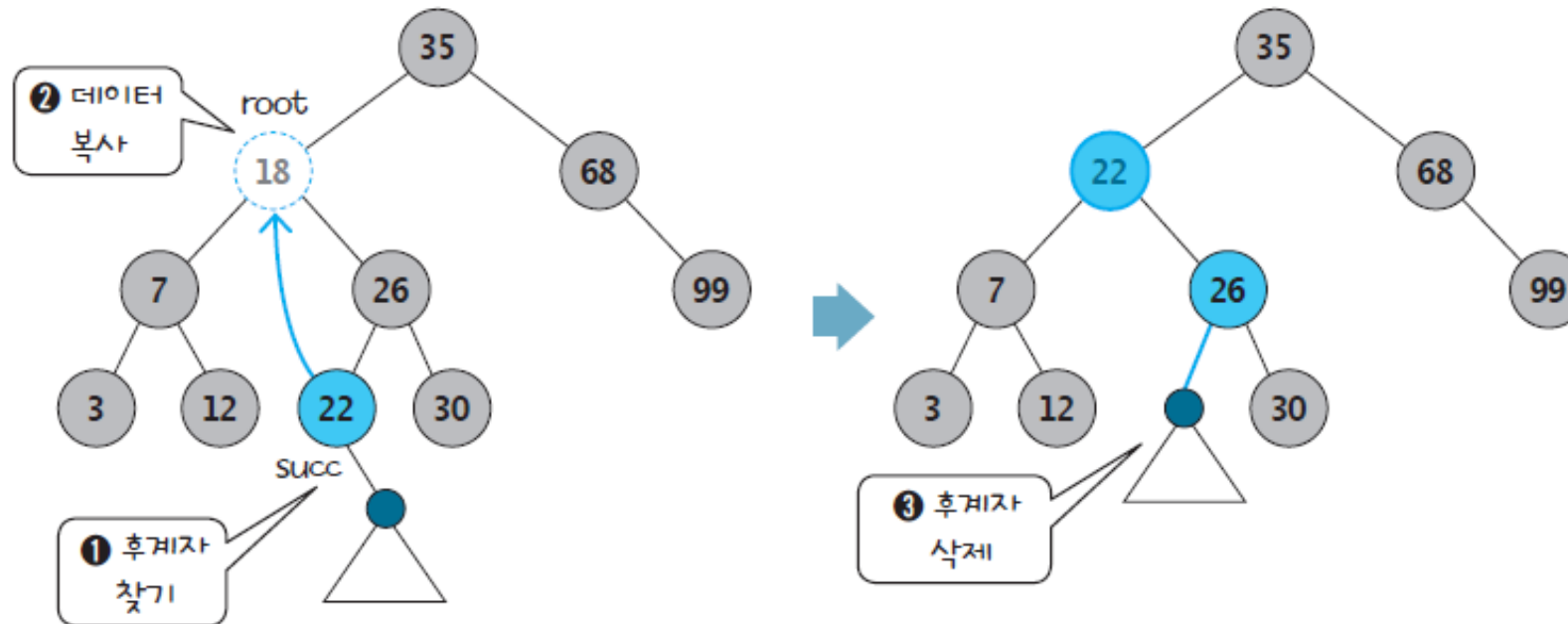
이진 탐색 트리 동작방식

- 이진 탐색 트리 동작 방식 : 삭제>Delete)
 - Case 3: 2개의 자식을 모두 갖는 노드의 삭제 (후계자 사용)



이진 탐색 트리 동작방식

- 이진 탐색 트리 동작 방식 - 삭제(Delete)
 - Case 3: 2개의 자식을 모두 갖는 노드의 삭제 (과정)



이진 탐색 트리 알고리즘

- 이진 탐색 트리(Binary Search Tree) 알고리즘

실습문제 : 이진 탐색 트리 알고리즘 구현하기

- 이진 탐색 트리 알고리즘을 파이썬으로 구현하세요.
 - 1) 이진 탐색 트리 알고리즘 (탐색 기능)
 - 2) 이진 탐색 트리 알고리즘 (삽입 기능)
 - 3) 이진 탐색 트리 알고리즘 (삭제 기능)

※ 가천대학교 컴퓨터공학과 '알고리즘' 과정

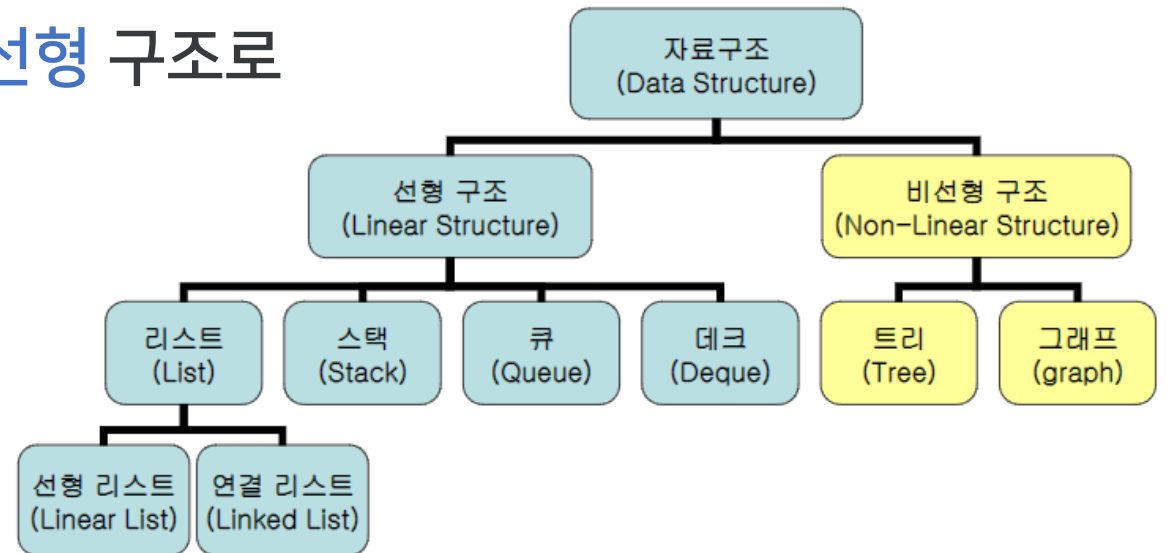
그래프 탐색

그래프 탐색

그래프 개념

■ 그래프 이론(Graph Theory)

- 그래프 이론(Graph Theory)은 수학의 한 분야
- 그래프(graph)라는 구조를 사용해 개체 간의 관계를 표현하고 분석하는 학문
- 여러 분야에서 복잡한 관계와 상호 작용을 시각화하고 이해하는 데 활용됨.
- 그래프는 자료 요소들의 관계가 비선형 구조로 나타날 때 사용되는 자료구조

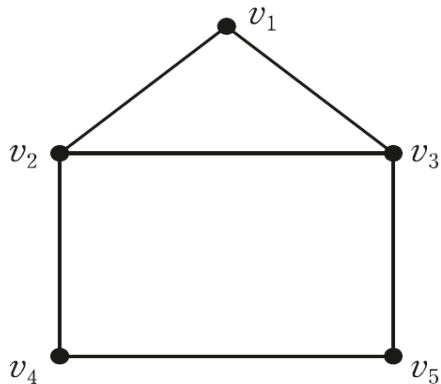


그래프 개념

■ 그래프(Graph)

- 그래프는 정점(Vertex, Node)과 간선(Edge, Arc)의 모음으로 구성되며, 이들을 사용해 여러 가지 문제를 모델링하고 해결할 수 있다.

그래프 graph G 는 순서쌍 (V, E) 로 정의한다. 여기서 $V = \{v_1, v_2, \dots, v_n\}$ 은 G 의 정점^{vertex} 혹은 노드^{node}의 집합이고, E 는 서로 다른 정점의 쌍 $\{v_i, v_j\}$ 의 집합이다. 이러한 쌍들을 간선^{edge}이라 한다.



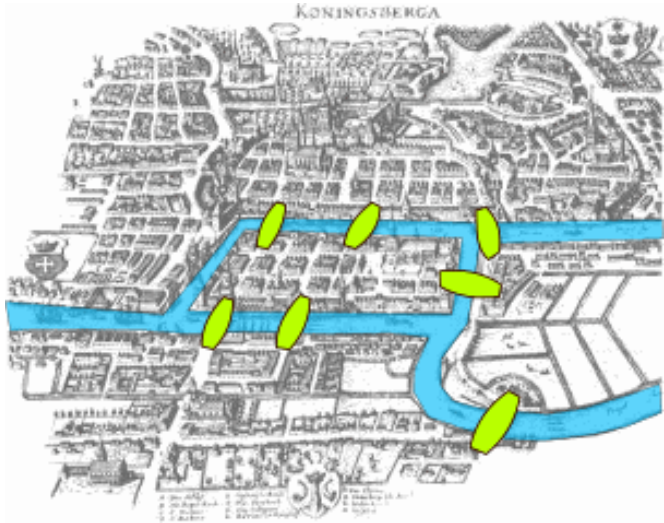
그래프 예

- 정점의 집합 $V = \{v_1, v_2, v_3, v_4, v_5\}$
 - 간선의 집합 $E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_5\}, \{v_4, v_5\}\}$
- 로 이루어진 그래프

그래프 이론의 발전

■ 그래프 이론의 기원

- 수학자 레온하르트 오일러(Leonhard Euler)가 1736년에 쓴 논문에서
 코니히스베르크의 다리 문제를 해결하면서 처음으로 그래프 이론의 기본 개념을 사용
- 코니히스베르크의 다리 문제:

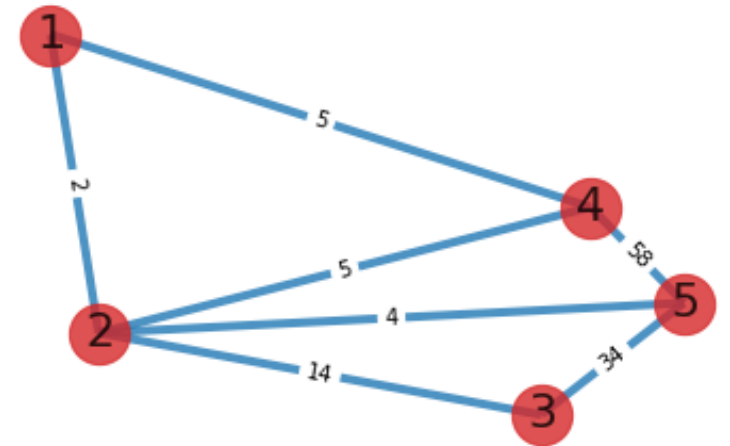
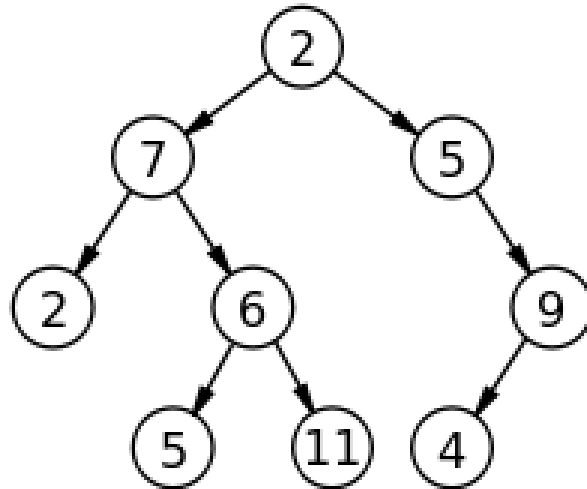
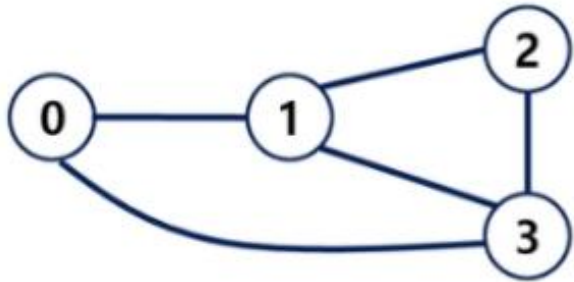


어느 정점에서 시작해서
모든 간선을 단 한 번 씩만
지나서 처음의 정점으로 올
수 있는 방법은?

그래프의 구성 요소

■ 그래프의 주요 구성 요소

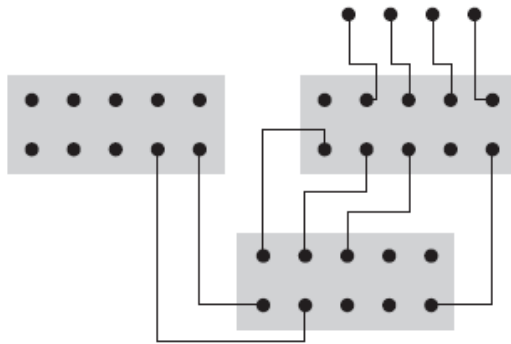
- 1) 노드(Node or Vertex) : 그래프에서 데이터를 나타내는 기본 단위
- 2) 간선(Edge or Arc) : 노드 간의 연결 관계를 나타내는 선(무방향: Edge, 유방향: Arc)
- 3) 가중치(Weight) : 엣지에 할당된 숫자 값, 두 노드 사이의 거리, 비용, 시간 등을 나타냄



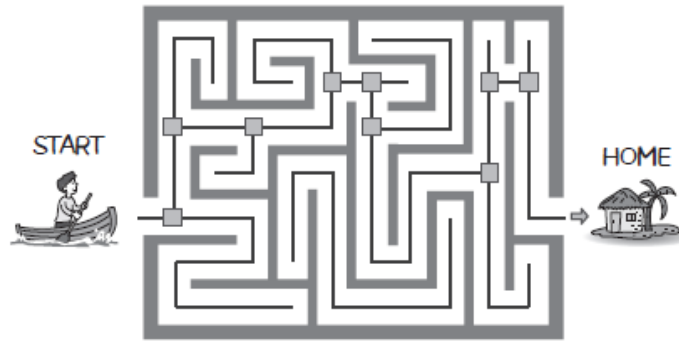
그래프 탐색

■ 그래프 탐색

- 그래프의 노드와 간선을 체계적으로 방문하여 정보나 경로를 찾는 알고리즘을 의미
- 탐색의 주된 목표는 특정 노드로부터 시작하여
모든 연결된 노드에 도달하거나, 특정 목표 노드에 도달하는 것



단자들 간의 연결성 검사



미로 탐색 문제

그래프 탐색 알고리즘

- 그래프 순회 알고리즘 예:

- 깊이 우선 탐색(DFS): 가능한 한 깊이 탐색한 후 백트래킹
- 너비 우선 탐색(BFS): 현재 위치에서 가까운 노드부터 탐색

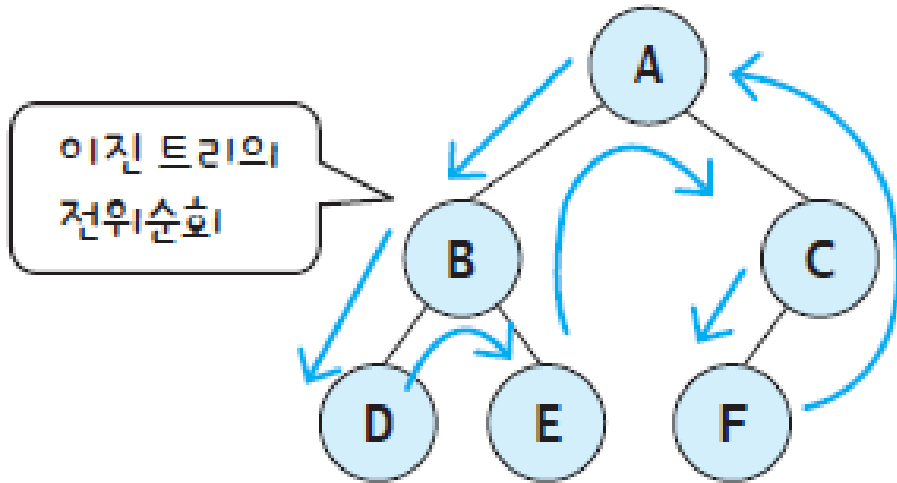
- 경로 탐색 알고리즘 예:

- 다익스트라 알고리즘: 최단 경로 탐색
- A* 알고리즘: 휴리스틱을 사용한 최단 경로 탐색

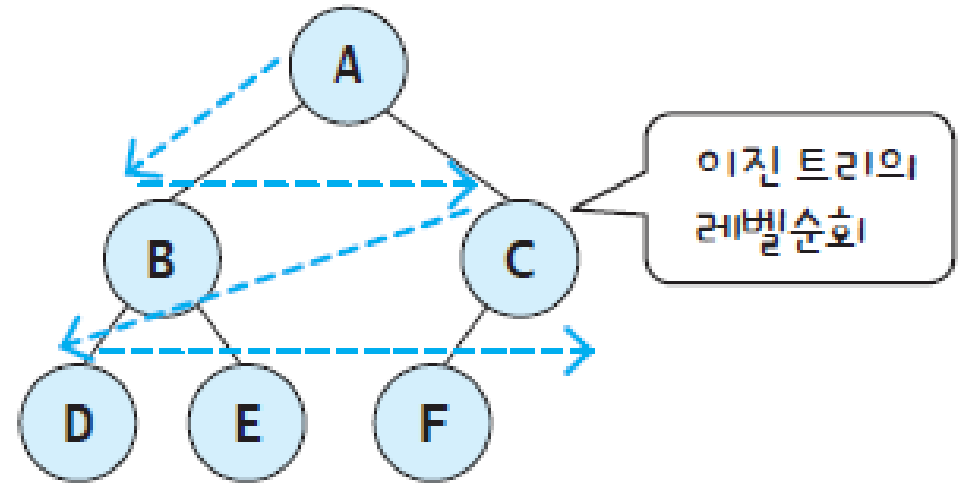
그래프 탐색 알고리즘

■ 대표적인 그래프 순회 알고리즘

- 깊이 우선 탐색(DFS: Depth-First Search)
- 너비 우선 탐색(BFS: Breadth-First Search)

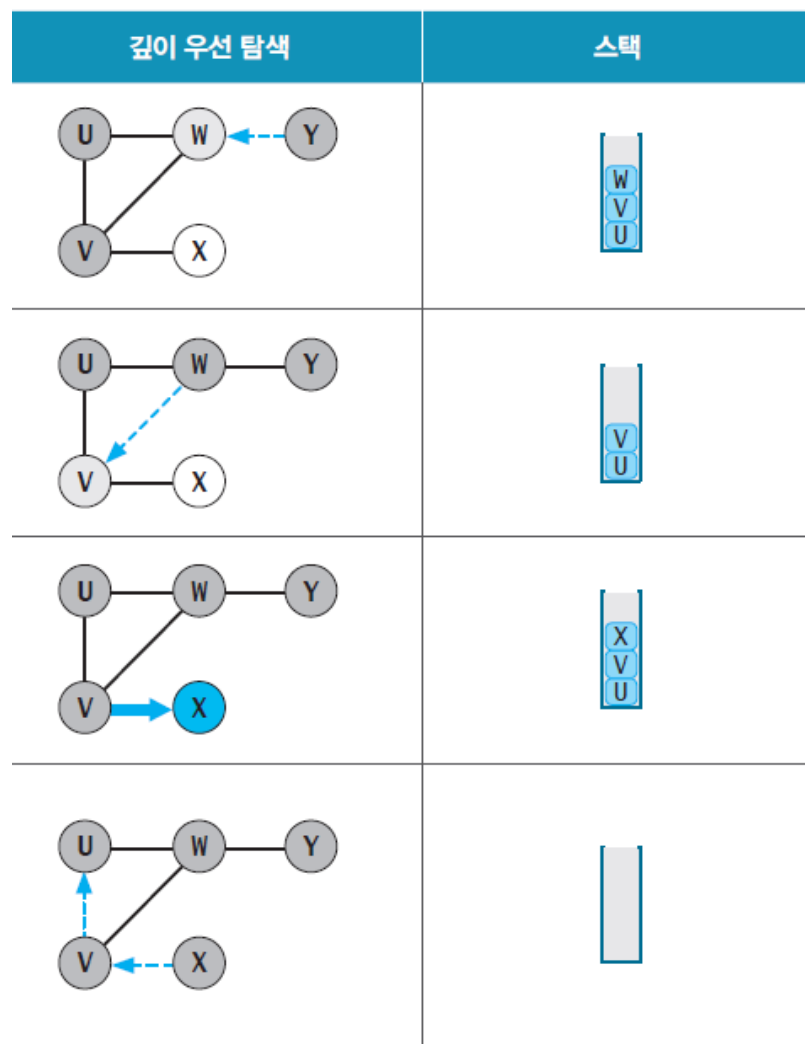
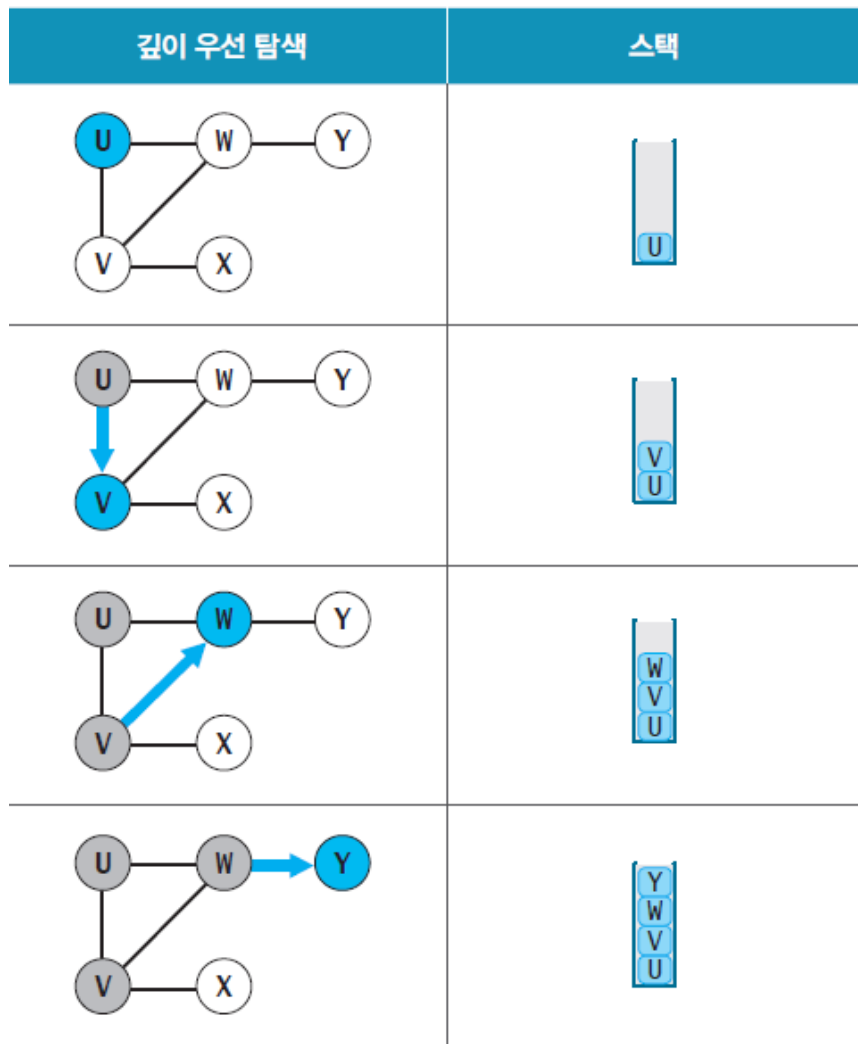


(a) 깊이 우선 탐색



(b) 너비 우선 탐색

깊이 우선 탐색



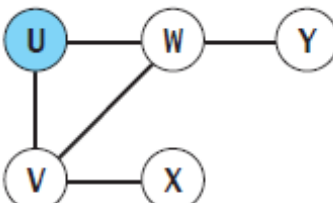

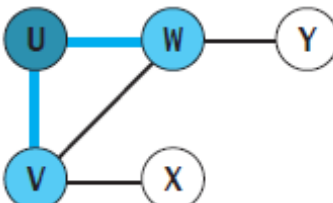

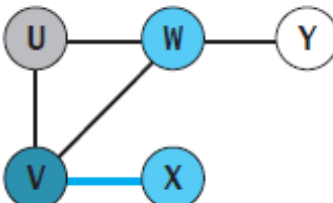
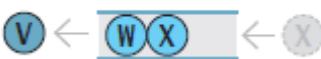
깊이 우선 탐색

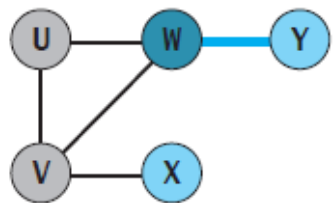

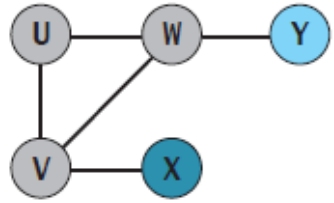

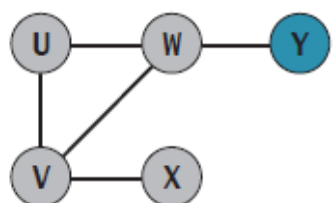

```
def dfs(vtx, adj, s, visited) :  
    print(vtx[s], end=' ')    # 현재 노드는 방문 했으므로, 화면에 출력하고  
    visited[s] = True        # True로 설정  
  
    for v in range(len(vtx)) :  
        if adj[s][v] != 0 :  
            if visited[v]==False:    # 방문하지 않은 이웃 노드V가 있으면  
                DFS(vtx, adj, v, visited)    # 그 노드를 시작으로 다시 DFS 호출
```

```
vtx = ['U', 'V', 'W', 'X', 'Y']  
adj_matrix = [[0, 1, 1, 0, 0],  
               [1, 0, 1, 1, 0],  
               [1, 1, 0, 0, 1],  
               [0, 1, 0, 0, 0],  
               [0, 0, 1, 0, 0]]
```

```
dfs(vtx, adj_matrix, 0, [False]*len(vtx))
```

너비 우선 탐색

너비 우선 탐색	큐
	
	
	

너비 우선 탐색

```
def bfs(adj_list, start):  
    """ 인접 리스트와 시작 노드를 받아 너비 우선 탐색을 수행하는 함수 """  
    visited = set()  
    queue = deque([start]) # 큐에 시작 노드 추가  
    visited.add(start)     # 방문 했다고 표시
```

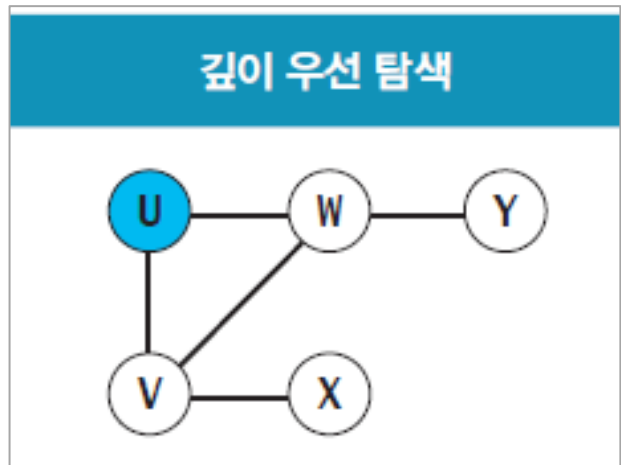
```
    while queue:  
        node = queue.popleft() # 큐의 왼쪽에서 꺼내기(dequeue)  
        print(node, end=' ')   # 노드 출력
```

```
        for neighbor in adj_list[node]: # 해당 노드의 인접 리스트에 있는 이웃 노드에 대해  
            if neighbor not in visited: # 방문하지 않은 노드라면  
                queue.append(neighbor) # 큐에 삽입하고  
                visited.add(neighbor)  # 방문 했다고 표시
```

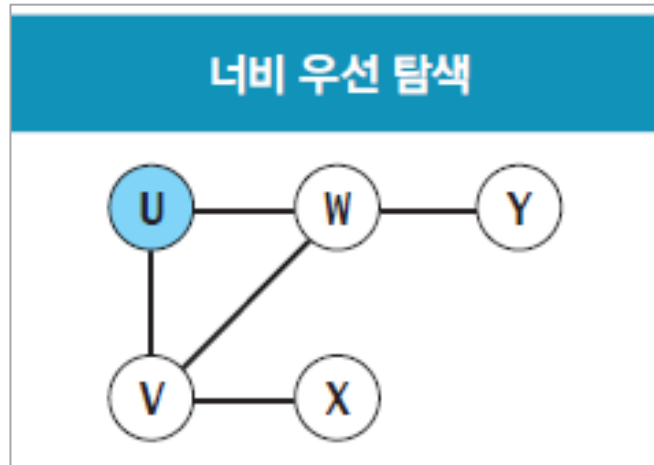
```
start_node = 'U'  
bfs(adj_list, start_node)
```

실습문제 : 그래프 탐색 구현하기

- 앞의 그래프 탐색 내용을 파이썬으로 구현하세요.
 - 1) 그래프의 깊이 우선 탐색(DFS)를 구현해 보세요.
 - 2) 그래프의 너비 우선 탐색(BFS)를 구현해 보세요



$U \rightarrow V \rightarrow W \rightarrow Y \rightarrow X$

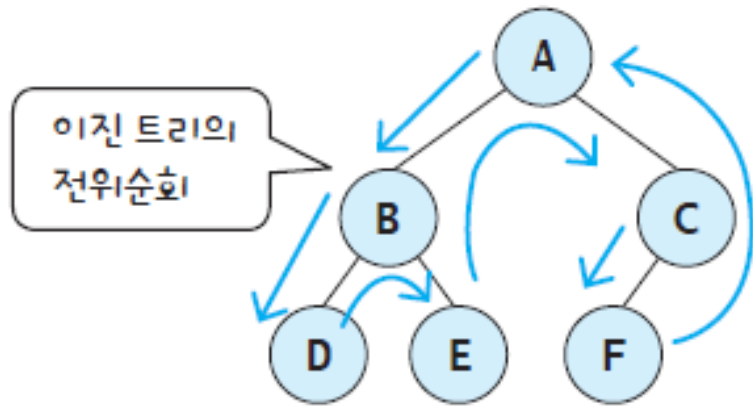


$U \rightarrow V \rightarrow W \rightarrow X \rightarrow Y$

실습문제 : 그래프 탐색 하기

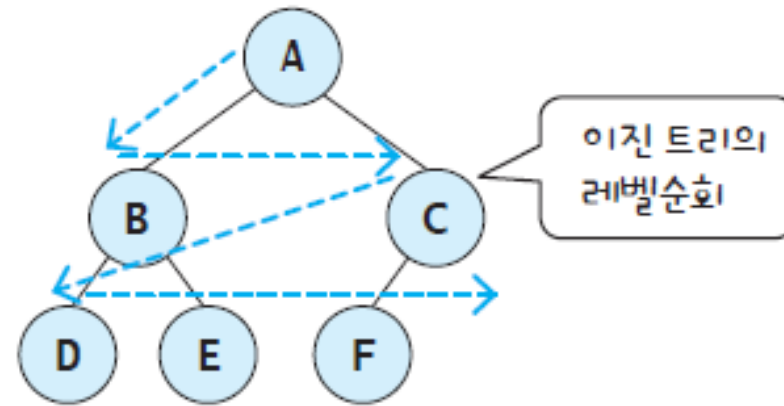
- 앞에서 만든 코드를 이용하여 그래프 탐색 결과 확인하기

- 1) 그래프의 깊이 우선 탐색(DFS) 결과
- 2) 그래프의 너비 우선 탐색(BFS) 결과



(a) 깊이 우선 탐색

$A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F$

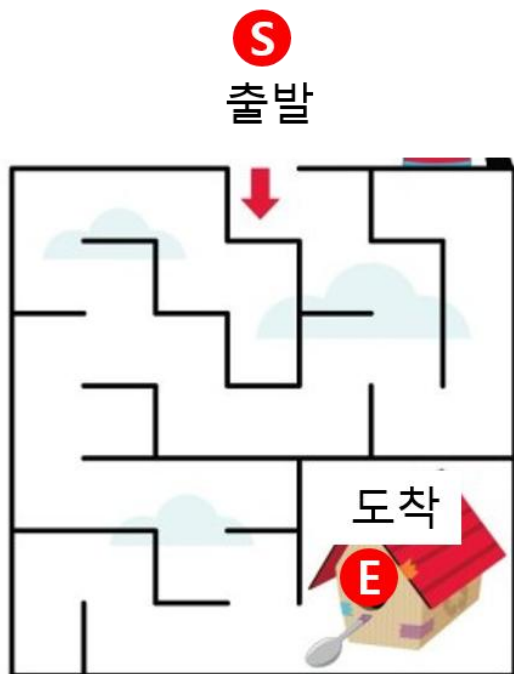


(b) 너비 우선 탐색

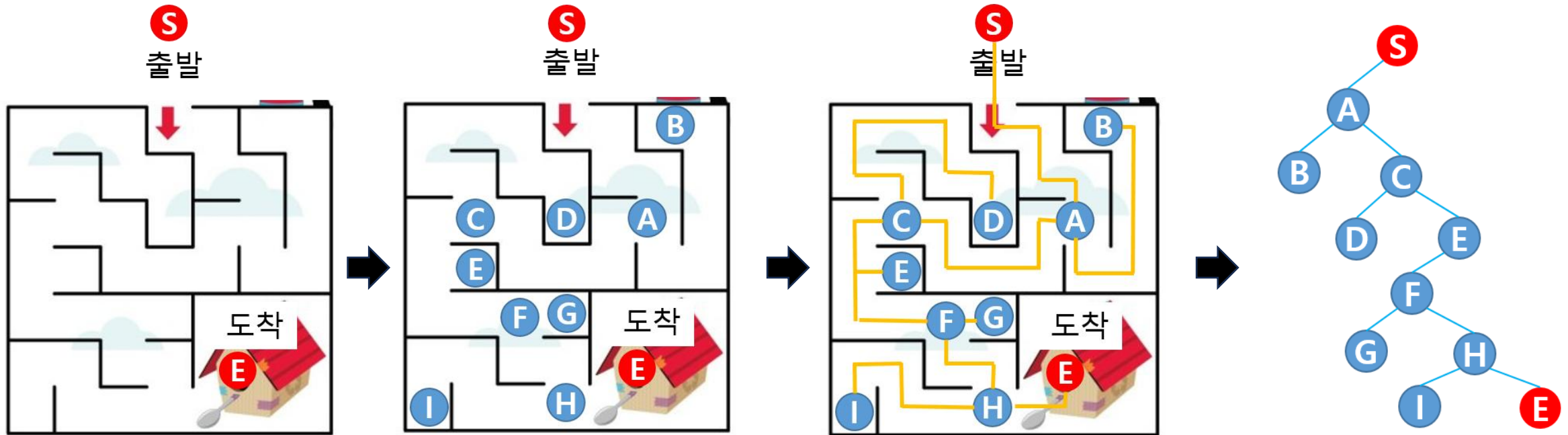
$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F$

실습문제 : 미로 탐색

- 아래 미로에 대해 그래프 탐색을 이용하여 도착 지점까지 경로를 탐색 하시오
 - 1) 그래프 경로 트리로 그려보기
 - 2) 그래프 깊이우선탐색/너비우선 탐색으로 그래프 탐색하기



실습문제 : 미로 탐색



1. '깊이 우선 탐색'

S →

E

2. '너비 우선 탐색'

S →

E

Q & A

Next Topic

- 알고리즘 중간고사

Keep learning, see you soon!