트리 1

-트리의 개념과 이진 트리-

HaRim Jung, Ph.D.
Visiting Professor / Senior Researcher
SKKU Institute for Convergence / Convergence Research Institute
Sungkyunkwan University, Korea

트리 (1/4)

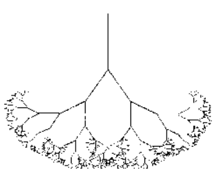
❑ 트리(Tree)의 정의

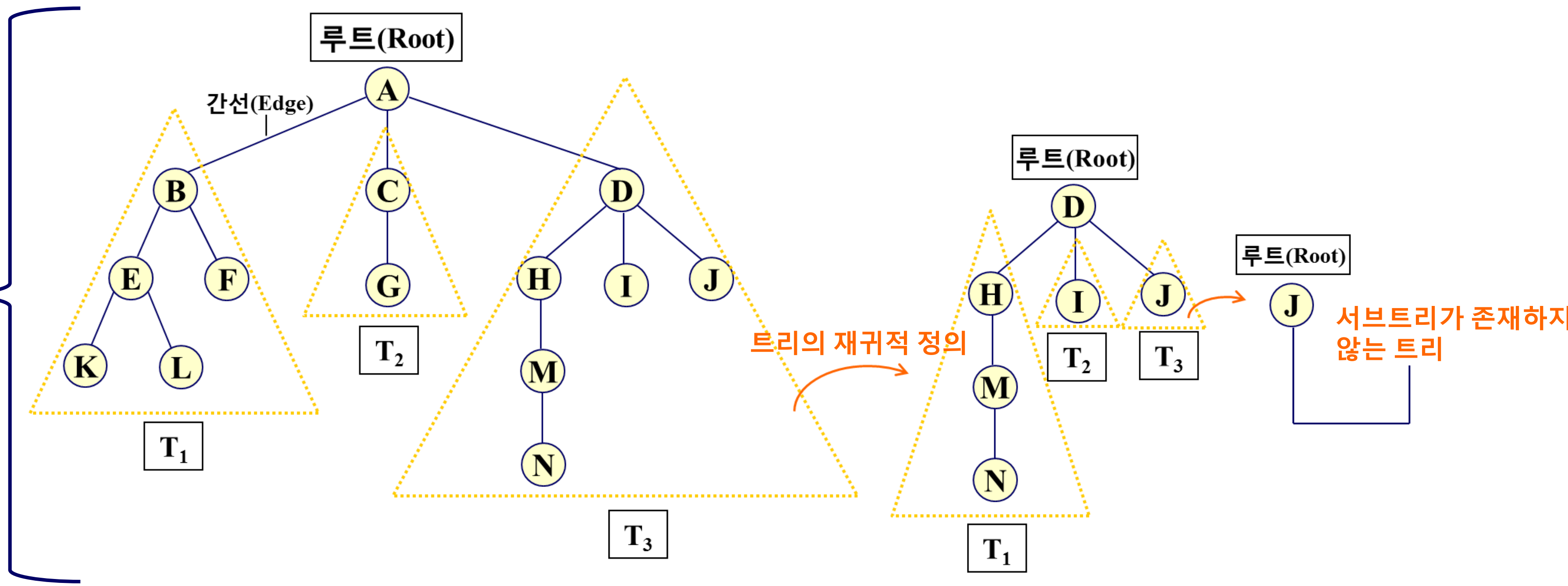
- **트리**는 아래의 조건을 만족하는 **하나 이상의 노드(Node)들로 구성된 유한 집합(Finite Set) T**, 여기서 노드는 트리의 기본 단위(사용자 정의 데이터 타입)

재귀적(Recursive) 정의

  – 노드들 중에는 **루트(Root)라 부르는 하나의 특별한 노드**가 존재함
  – 루트를 제외한 나머지 노드들은 원소가 중복되지 않는 $N(\geq 0)$ 개의 부분 집합 $T_1, T_2, \ldots, T_N$으로 나누어지며 $T_i(1 \leq i \leq N)$는 하나의 트리임, 이 때 각 T_i를 트리 T의 **서브 트리(Subtree)**라고 부름

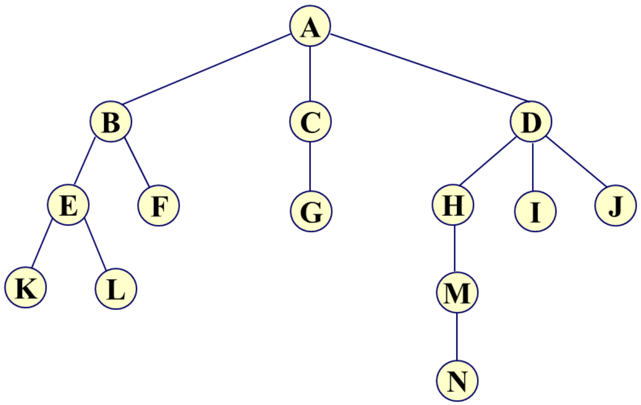
나무를 거꾸로 세워 놓은 형태의 자료구조





트리 (2/4)

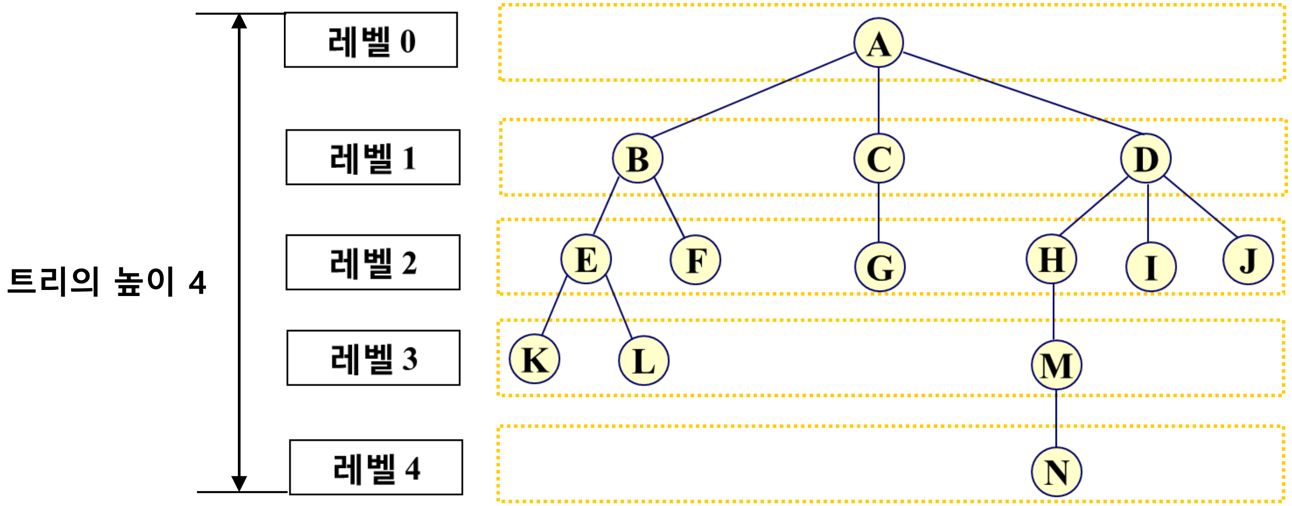
❑ 트리의 용어(Terminology)



- 트리의 구성 요소에 해당하는 A, B, C, ..., N을 **노드(Node)**라 함
- (1) 노드 B와 간선으로 연결되어 있으며 (2) B 바로 아래에 위치하는 노드 E, F를 각각 B의 **자식 노드(Child Node)**라 함
- (1) 노드 E, F와 간선으로 연결되어 있으며 (2) E, F 바로 위에 위치하는 노드 B를 E, F의 **부모 노드(Parent Node)**라 함
- 특정 노드의 자식 노드 수를 **차수(Degree)**라고 함
- 트리에 존재하는 모든 노드들 중 차수 값이 가장 큰 노드의 차수를 **트리의 차수(Dgree of Tree)**라고 함
- 동일한 부모 노드 B를 가지는 노드 E, F를 서로 **남매 혹은 형제 혹은 자매 노드(Sibling Node)**라 함
- 부모가 없는 노드(i.e., 트리의 최상위 노드) A를 **루트 혹은 루트 노드 (Root Node)**라 함
- 자식이 없는 노드 K, L, F, G, N, I, J를 각각 **단말 노드(Leaf Node)**라 함
- 단말 노드를 제외한 모든 노드들을 각각 **내부 노드(Internal Node)** 혹은 **비단말 노드(Non-leaf Node)**라 함
- 노드 B, K를 포함하여 B, K 사이에 간선으로 연결된 노드들의 리스트 B, E, K(혹은 K, E, B)를 B에서부터 K까지(혹은 K에서부터 B까지)의 **경로(Path)**라 함 (**NOTE:** **트리에서 경로에 속한 노드는 중복되지 않음**, e.g., B, F, B, E, K는 B에서부터 K까지의 경로가 아님)
- 노드 K로부터 루트 노드 A까지의 경로 상에 존재하는 노드들 중 K를 제외한 노드 E, B, A를 각각 K의 **조상 노드(Ancestor Node)**라 함

트리 (3/4)

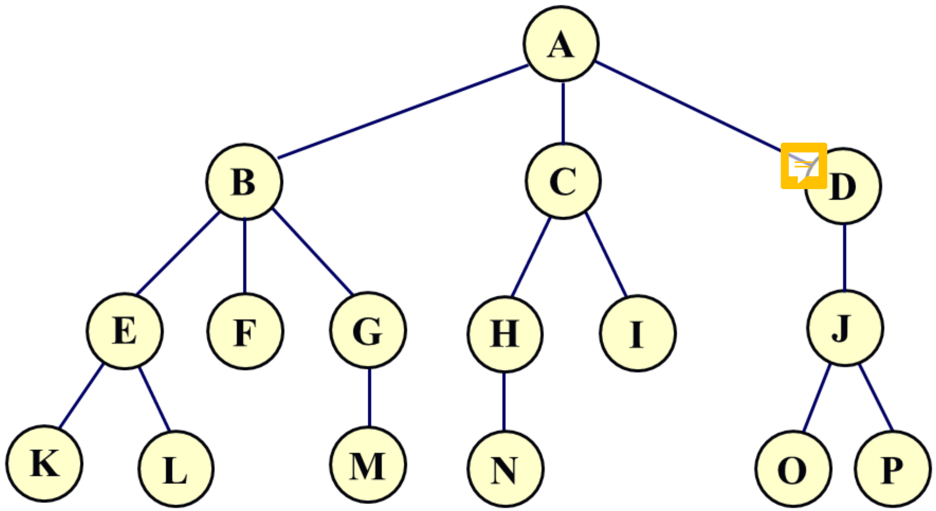
❑ 트리의 용어(Terminology) contd.



- 트리 T의 부분 집합을 이루는 트리 T_i를 T의 **서브 트리(Subtree)**라고 함
  - **T의 서브트리 T_i는** T의 루트 노드를 제외한 임의의 노드를 루트 노드로 하는 **트리임** (**NOTE**: 트리의 (재귀적) 정의에 의해 노드 L은 노드 L을 루트 노드로 하고 서브 트리가 존재하지 않는 트리임)
- 특정 노드 B를 루트로 하는 트리에서 B를 제외한 모든 노드 E, F, K, L을 B의 **후손 노드(Descendant Node)**라 함
- 임의의 노드로부터 루트 노드까지의 경로 상에 존재하는 간선의 수를 **레벨(Level)** 혹은 **깊이(Depth)**라 함
  - 예: 노드 K의 레벨은 3, 노드 E의 레벨은 2, 노드 B의 레벨은 1, 루트 노드 A의 레벨은 0
  - **NOTE:** 루트 노드를 레벨 1로 하고 하위 단계로 내려갈 때마다 차례로 레벨을 증가시키는 정의(예: 노드 K의 레벨은 4, 노드 E의 레벨은 3, 노드 B의 레벨은 2, 루트 노드 A의 레벨은 1)도 있지만 위의 정의가 더 일반적임
- 트리에 존재하는 모든 노드들 중 레벨 값이 가장 큰 노드의 레벨을 **트리의 높이(Height of Tree)**라고 함
- 노드에 저장된 정보 중 탐색 시 사용되는 정보를 **키(Key)**라고 함

트리 (4/4)

❑ 트리의 용어(Terminology) contd.

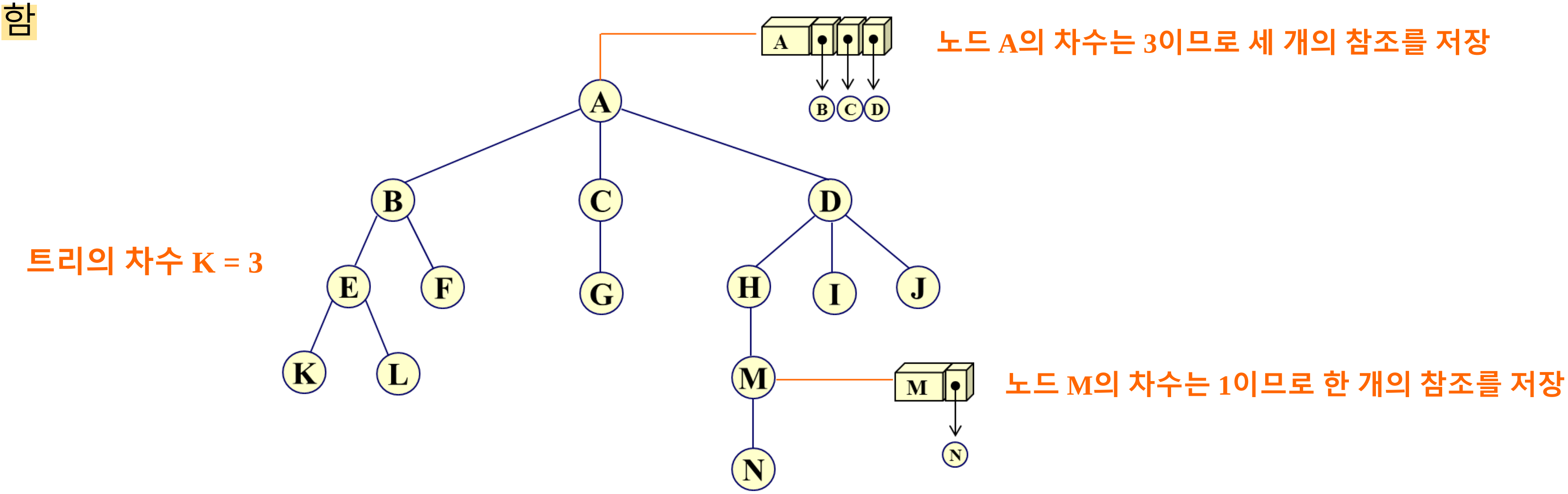


- 루트 노드는?
- 노드 A의 자식 노드는?
- 노드 A의 차수는?
- 노드 B, C, D의 부모 노드는?
- 단말 노드는?
- 노드 C의 자손 노드는?
- 노드 P의 조상 노드는?
- 노드 E의 레벨은?
- 트리의 높이는?
- 트리의 차수는?

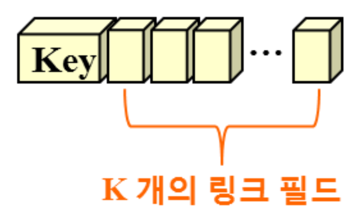
트리의 왼쪽 자식-오른쪽 남매 표현 (1/3)

❑ 왼쪽 자식-오른쪽 남매 (Left Child-Right Sibling) 표현

- 트리를 메모리에 저장하기 위해서는 트리의 각 노드 N에 대해 N의 키와 자식 수만큼의 참조(Reference)를 저장해야 함



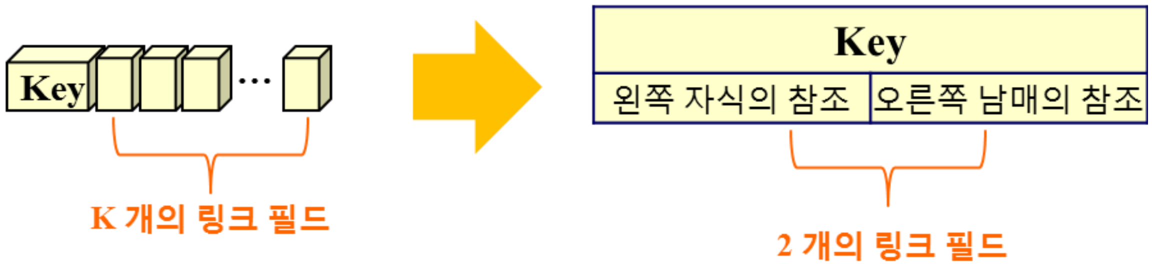
  – 따라서 N은 가변적인 수의 참조를 저장하게 되지만 물리적 구현의 단순성을 위해서 모든 노드가 트리의 차수 K만큼의 참조를 저장할 수 있도록 하는 것이 바람직함
  – 하지만 차수가 K인 트리 T에 존재하는 노드의 수가 N개일 때, None 값을 저장하고 있는 링크 필드의 수(i.e., 낭비 되는 링크 필드의 수)는 N·K – (N – 1)임, 여기서 N·K는 T에 존재하는 모든 링크 필드의 수이고 (N-1)은 T에서 부모 노드와 자식 노드를 연결하는 참조의 수
    - K 값이 클수록 메모리의 낭비가 심해지는 것은 물론 트리를 탐색하는 과정에서 참조가 None인 링크 필드도 확인해야 하므로 시간적으로도 매우 비효율적임(단, 메모리 기반 트리에 한함)
    - 예를 들어, 위의 트리는 29 개의 링크 필드가 None 값으로 채워지게 됨



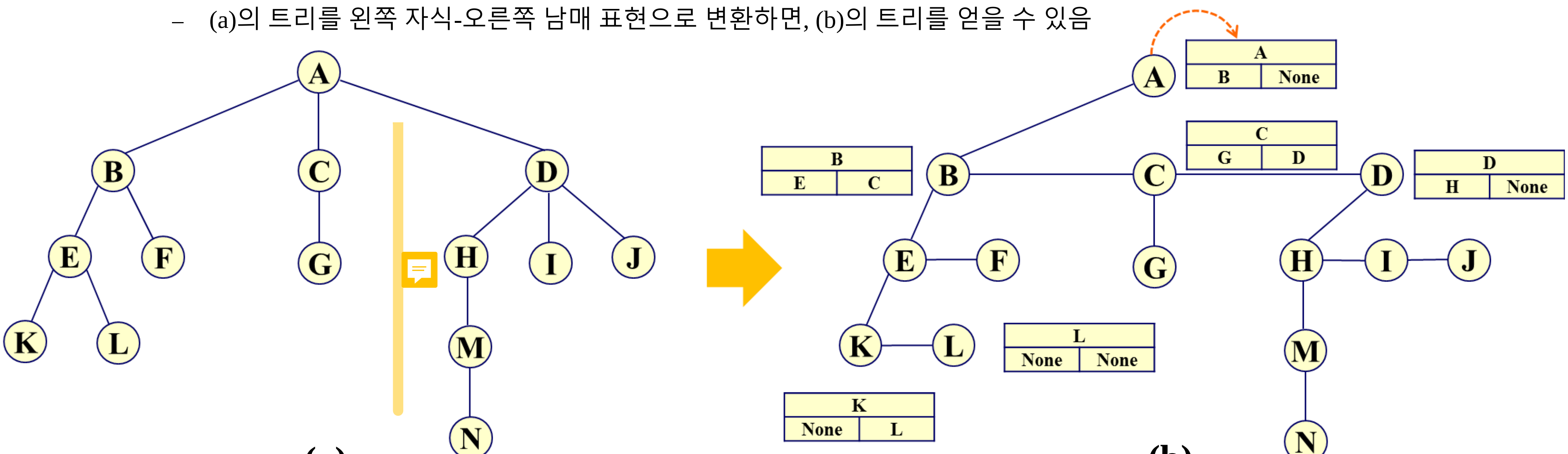
트리의 왼쪽 자식-오른쪽 남매 표현 (2/3)

❑ 왼쪽 자식-오른쪽 남매 (Left Child-Right Sibling) 표현 contd.

- **왼쪽 자식-오른쪽 남매 표현 방법**은 **K의 값을 2로 제한**하여 트리를 표현할 수 있도록 만드는 방법임
  - 각 노드는 자신의 제일 왼쪽 자식(자식이 하나면 그 자식)과 자신의 오른쪽 남매에 대한 두 개의 참조를 저장하기 위한 두 개의 링크 필드만을 가짐



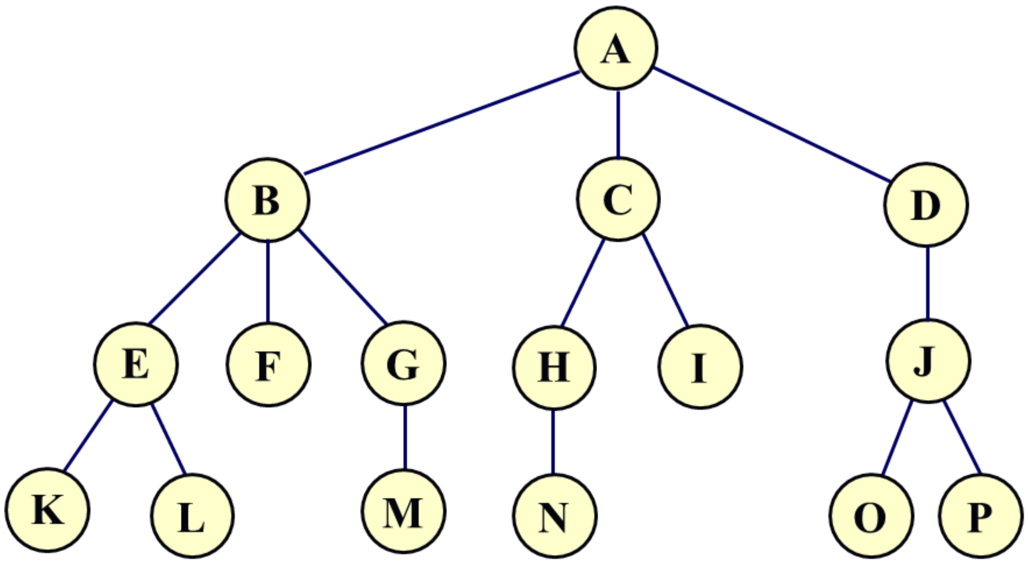
  - (a)의 트리를 왼쪽 자식-오른쪽 남매 표현으로 변환하면, (b)의 트리를 얻을 수 있음



(a) (b)

트리의 왼쪽 자식-오른쪽 남매 표현 (3/3)

- **왼쪽 자식-오른쪽 남매 (Left Child-Right Sibling) 표현** contd.
  - **연습:** 다음의 트리를 왼쪽 자식-오른쪽 남매로 표현하시오.

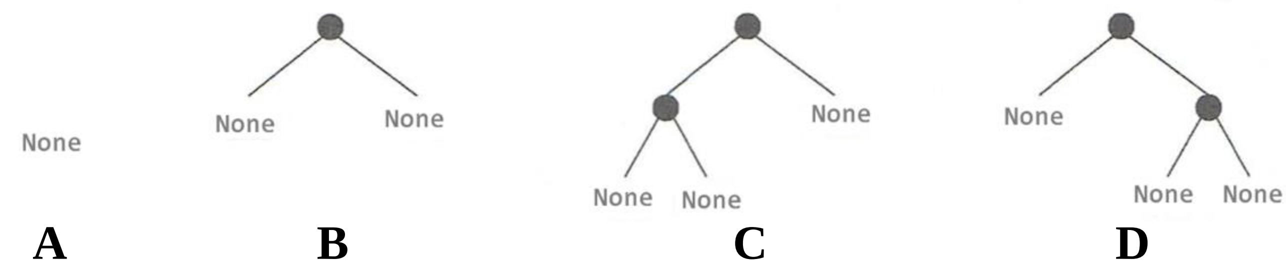


이진 트리(1/3)

❑ 이진 트리(Binary Tree)의 정의

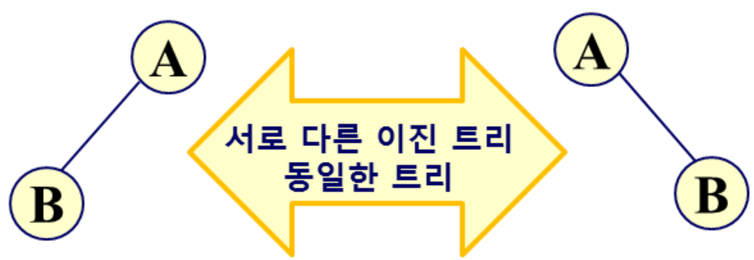
- **이진 트리**는 empty이거나, empty가 아니면 루트 노드와 두 개의 이진 트리인 왼쪽 서브 트리와 오른쪽 서브 트리로 구성된 트리임(**재귀적 정의**)

A. empty 이진 트리
B. 루트 노드만 있는 이진트리
C. 루트 노드의 오른쪽 서브 트리가 empty인 이진 트리
D. 루트 노드의 왼쪽 서브 트리가 empty인 이진 트리



❑ 이진 트리의 특징

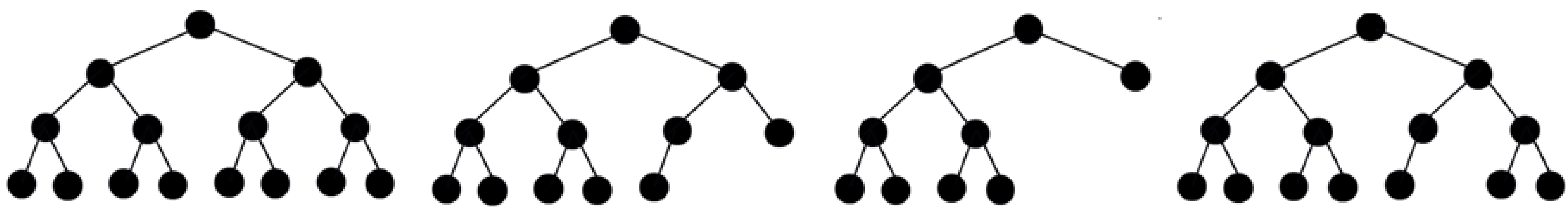
- 이진 트리는 모든 노드의 차수가 2를 넘지 않음
  - 따라서 임의의 노드는 **최대 두 개의 자식 노드**를 가질 수 있음
- 자식 노드들의 순서를 구별하지 않는 트리와는 다르게 이진 트리는 **자식 노드의 순서를 구별**함
- **이진 트리와 차수가 2인 트리와의 차이점**
  - 이진 트리는 empty인 트리가 존재하지만, 차수가 2인 트리(by Definition)는 empty인 트리가 존재하지 않음
  - 이진 트리는 서브 트리들의 순서(i.e., 자식 노드들의 순서)를 구별하지만, 차수가 2인 트리는 서브 트리들의 순서를 구별하지 않음



이진 트리(2/3)

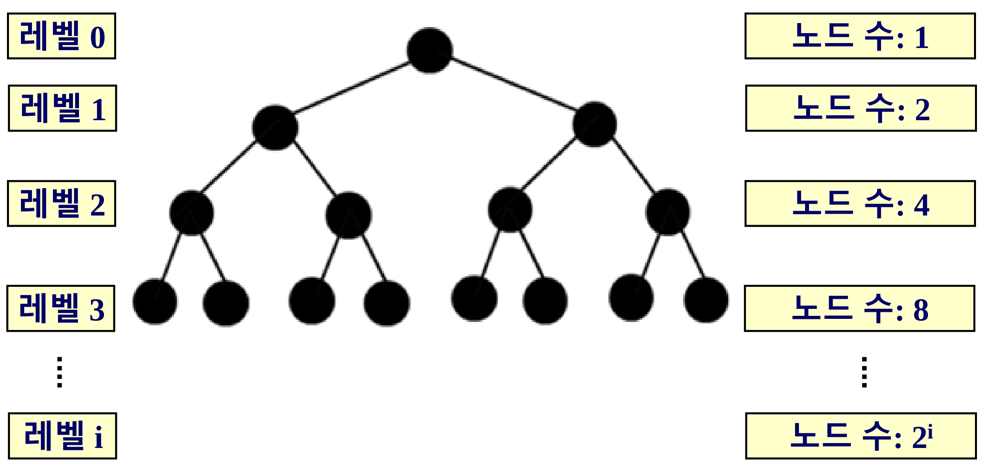
❑ 이진 트리의 특징 contd.

- **특별한 형태의 이진 트리**
  - **완벽 이진 트리(Perfect Binary Tree):** 각 내부 노드가 두 개의 자식 노드를 가지며 모든 단말 노드의 레벨이 동일한 이진 트리
  - **완전 이진 트리(Complete Binary Tree)**: 마지막 레벨을 제외한 각 레벨이 노드들로 꽉 차있고, 마지막 레벨에는 노드들이 왼쪽부터 빠짐없이 채워진 트리(**NOTE:** 완벽 이진 트리라면 완전 이진 트리, 역은 성립하지 않음)



(a) 완벽 이진 트리 **(b) 완전 이진 트리** **(c) 불완전한 이진 트리** **(d) 불완전한 이진 트리**

- 이진 트리의 레벨 i에 존재할 수 있는 최대 노드의 수는 2^i (i = 0, 1, 2, 3,...)



이진 트리의 레벨 i에 존재할 수 있는 최대 노드의 수는
완벽 이진 트리인 경우 레벨 i에 존재하는 노드의 수

이진 트리(3/3)

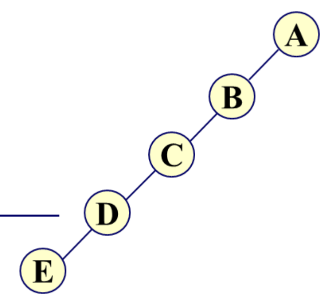
❑ 이진 트리의 특징 contd.

- 높이가 h인 완벽(Perfect) 이진 트리에 존재하는 모든 노드의 수 N은 **$2^{h+1} - 1$** (i = 0, 1, 2, 3,...)
  - 레벨 0에 존재하는 노드의 수는 1, 레벨 1에 존재하는 노드의 수는 2, 레벨 2에 존재하는 노드의 수는 4,..., 마지막 레벨 h에 존재하는 노드의 수는 2^h (**NOTE**: 트리에 존재하는 모든 노드들 중 레벨 값이 가장 큰 노드의 레벨, i.e., 마지막 레벨이 트리의 높이 h)
  - 따라서 높이가 h인 완벽 이진 트리에 존재하는 모든 노드의 수 $N = 2^0 + 2^1 + 2^2 +, ..., + 2^h = 2^{h+1} - 1$

Proof:

$$\begin{aligned} 2 \cdot N &= \phantom{2^0 + {}} 2^1 + 2^2 +, ..., + 2^h + 2^{h+1} \\ N &= 2^0 + 2^1 + 2^2 +, ..., + 2^h \\ \hline 2 \cdot N - N &= 2^{h+1} - 1 \\ N &= 2^{h+1} - 1 \end{aligned}$$

- 완벽 이진 트리에 존재하는 노드의 수가 N이라면 해당 트리의 높이 h = **$\log_2(N+1) - 1$**
- 높이가 h인 완전(Complete) 이진 트리에 존재할 수 있는 노드의 수 N은 **$2^h \le N \le 2^{h+1} - 1$**
  - **N이 2^h보다 작으면 높이가 h-1이 되고**(**NOTE:** 높이가 h-1인 완벽 이진 트리에 존재하는 모든 노드의 수 $N = 2^h - 1$), **N이 $2^{h+1} - 1$보다 크면 높이가 h+1이 되기 때문**
- 완전 이진 트리에 존재하는 노드의 수가 N이라면 해당 트리의 높이 h = **$\lceil \log_2(N+1) \rceil - 1$**
- 이진 트리에 존재하는 노드의 수가 N일 때 해당 트리의 최대 높이 $\boxed{h = N - 1}$



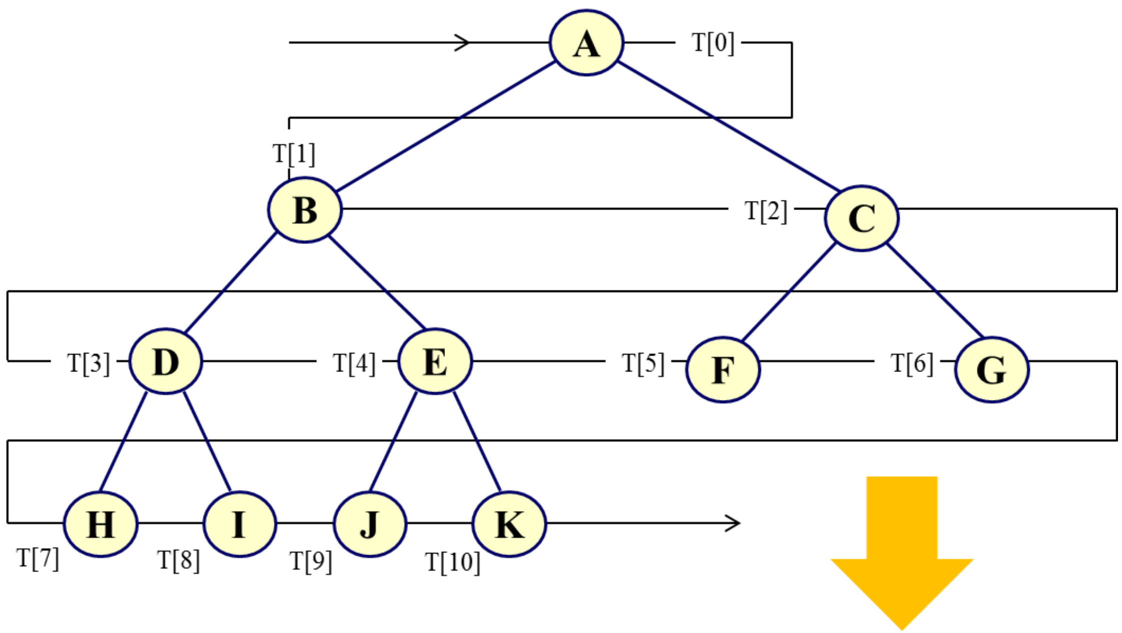
이진 트리의 구현(1/13)

❑ 이진 트리의 (물리적) 구현

- Python 리스트(동적 배열)를 이용한 표현(구현)과 노드들의 참조를 이용하여 연결 시키는(연결 리스트의 변형) 구현

❑ Python 리스트를 이용한 표현

- empty인 Python 리스트 T를 생성 후, 아래 그림과 같이 레벨 0(루트 노드)부터 마지막 레벨 순서로 내려가며, 각 레벨에서는 좌에서 우로 트리의 노드들을 append(item)을 이용하여 T[0]부터 차례로 저장

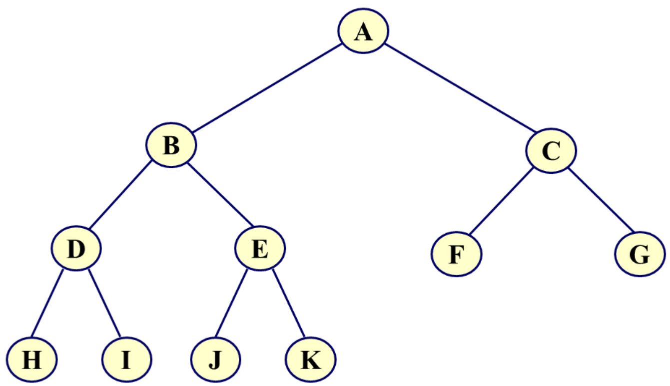


INDEX:	0	1	2	3	4	5	6	7	8	9	10
	A	B	C	D	E	F	G	H	I	J	K

이진 트리의 구현(2/13)

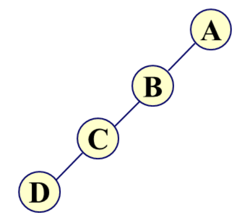
❑ Python 리스트를 이용한 표현 contd.

- Python 리스트를 이용하여 표현하면 트리 T에서 특정 노드의 부모 노드와 자식 노드가 어디에 저장되어 있는지(i.e., Python 리스트의 **인덱스 값**)를 다음과 같은 규칙을 통해 쉽게 알 수 있음
  - T[i]에 저장되어 있는 노드의 왼쪽 자식 노드는 **T[2·i + 1]**에 위치
  - T[i]에 저장되어 있는 노드의 오른쪽 자식 노드는 **T[2·(i + 1)]**에 위치
  - T[i]에 저장되어 있는 노드의 부모는 **T[(i - 1) // 2]**에 위치(단, i > 0)



INDEX:	0	1	2	3	4	5	6	7	8	9	10
	A	B	C	D	E	F	G	H	I	J	K

완전 이진 트리의 경우



INDEX:	0	1	2	3	4	5	6	7
	A	B		C				D

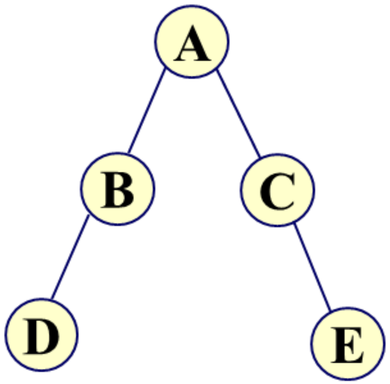
편향 이진 트리의 경우

- Python 리스트를 이용하여 편향 이진 트리(Skewed Binary Tree)를 표현할 경우 트리의 높이가 커질 수록 메모리 낭비가 심화됨 (**NOTE:** 완전 이진 트리를 구현할 경우에는 낭비되는 메모리 공간이 전혀 없음)

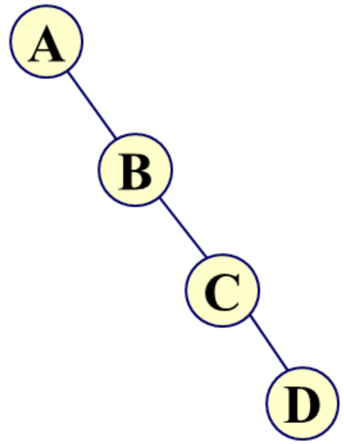
이진 트리의 구현(3/13)

❑ Python 리스트를 이용한 표현 contd.

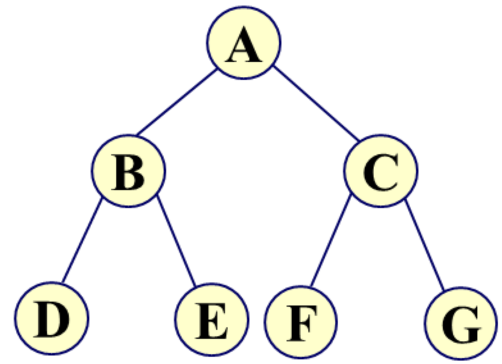
- **연습:** 다음 이진 트리를 Python 리스트로 표현하시오.



(a)



(b)



(c)

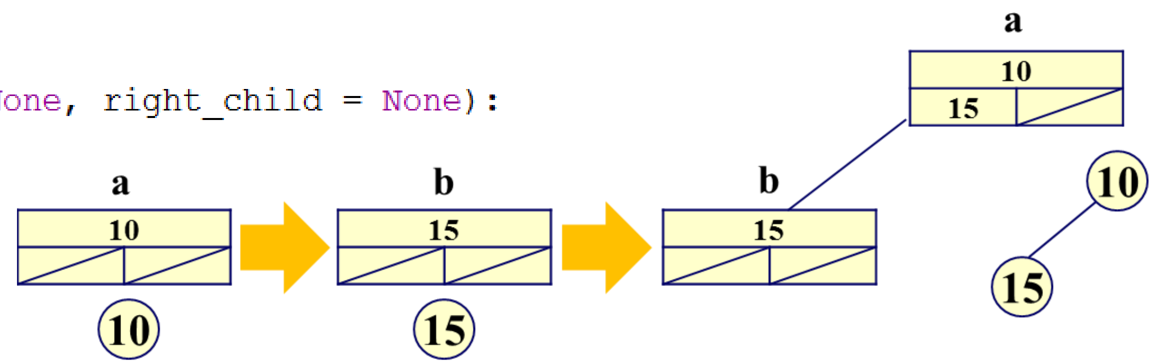
이진 트리의 구현(4/13)

❑ 노드들의 참조를 이용하여 연결 시키는 구현

- **(이진) 트리를 구성하는 단위: 노드**
  - 일반적인 경우의 이진 트리의 노드는 **키(항목 값)**, 왼쪽 자식 노드의 참조를 저장하기 위한 **left 링크 필드**, 오른쪽 자식 노드의 참조를 저장하기 위한 **right 링크 필드**로 구성됨

key	
left	right

  - **노드 객체를 위한 Node 클래스 정의**




```python
class Node:
    def __init__(self, item, left_child = None, right_child = None):
        self.key = item
        self.left = left_child
        self.right = right_child
    def get_key(self): return self.key
    def get_left(self): return self.left
    def get_right(self): return self.right

    def set_key(self, new_item):
        self.key = new_item
    def set_left(self, new_left_child):
        self.left = new_left_child
    def set_right(self, new_right_child):
        self.right = new_right_child
```

Node 클래스

```python
if __name__ == "__main__":
    a = Node(10)
    print(a.get_key())
    print(a.get_left())
    print(a.get_right())
    b = Node(15)
    a.set_left(b)
    print(a.get_left().get_key())
```

일련의 연산과 출력

```
10
None
None
15
```

결과

이진 트리의 구현(5/13)

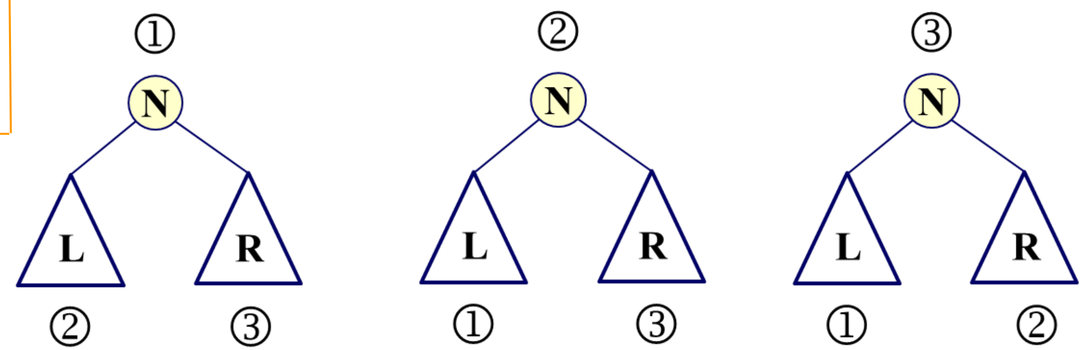
❑ 노드들의 참조를 이용하여 연결 시키는 구현 contd.

- 이진 트리에 적용 가능한 주요 연산
  - **BinaryTree():** 빈 이진 트리 생성
  - **preorder(node):** 이진 트리를 전위 순회
  - **inorder(node):** 이진 트리를 중위 순회
  - **postorder(node):** 이진 트리를 후위 순회
  - **levelorder(root):** 이진 트리를 레벨 순회

⋮

순회(Traversal)

- 이진 트리의 연산 중 가장 기본적인 연산
- 각각 방식은 다르지만 순회는 모두 루트 노드부터 시작
- 대부분의 이진 트리의 연산은 트리를 순회하며 이루어짐



전위순회: NLR **중위순회: LNR** **후위순회: LRN**

-순회-

- **전위, 중위, 후위 순회는 모두 루트로부터 동일한 순서(깊이 우선)로 이진 트리의 노드를 지나가는데, 특정 노드 N에 도착 했을 때 N을 방문(i.e., N에 대한 작업을 수행)하는지, 일단 지나치고 나중에 방문하는지에 따라 구분됨**
  - **전위 순회**는 (1) N에 도착하면 N을 먼저 방문, 그 후 (2) N의 왼쪽 자식 노드를 루트로 하는 서브 트리 T1의 모든 노드를 방문(**T1을 순회함으로써**), 마지막으로 (3) N의 오른쪽 자식 노드를 루트로 하는 서브 트리 T2의 모든 노드를 방문(**T2를 순회함으로써**)
  - **중위순회**는 N에 도착하면 N의 방문을 보류하고 (1) N의 왼쪽 자식 노드를 루트로 하는 서브 트리의 모든 노드를 방문, 그 후 (2) N을 방문, 마지막으로 (3) N의 오른쪽 자식 노드를 루트로 하는 서브 트리의 모든 노드를 방문
  - **후위순회**는 N에 도착하면 N의 방문을 보류하고 (1) N 왼쪽 자식 노드를 루트로 하는 서브 트리의 모든 노드를 방문, 그 후 (2) n의 오른쪽 자식 노드를 루트로 하는 서브 트리의 모든 노드를 방문, 마지막으로 (3) N을 방문
- **레벨 순회**는 루트 노드가 있는 최상위 레벨부터 시작하여 각 레벨마다 좌에서 우로(**너비 우선**) 노드를 방문

이진 트리의 구현(6/13)

❑ 노드들의 참조를 이용하여 연결 시키는 구현 contd.

- **이진 트리 객체를 위한 클래스 정의**

```
from btnode import Node
from clqueue import Queue

class BinaryTree:
    def __init__(self):
        self.root = None

    def preorder(self, node):
        if node != None:
            print(str(node.get_key()),' ', end='')
            if node.get_left():
                self.preorder(node.get_left())
            if node.get_right():
                self.preorder(node.get_right())

    def inorder(self, node):
        if node != None:
            if node.get_left():
                self.inorder(node.get_left())
            print(str(node.get_key()),' ', end='')
            if node.get_right():
                self.inorder(node.get_right())
```

계속

```
    def postorder(self, node):
        if node != None:
            if node.get_left():
                self.postorder(node.get_left())
            if node.get_right():
                self.postorder(node.get_right())
            print(str(node.get_key()),' ', end='')

    def levelorder(self, root):
        q = Queue()
        q.enqueue(root)
        while not q.is_empty():
            node = q.dequeue()
            print(str(node.get_key()), ' ', end='')
            if node.get_left():
                q.enqueue(node.get_left())
            if node.get_right():
                q.enqueue(node.get_right())
```

이진 트리의 구현(7/13)

❑ 노드들의 참조를 이용하여 연결 시키는 구현 contd.

- **BinaryTree()** **시간복잡도: O(1)**

```
5     def __init__(self):
6         self.root = None
```

  – 빈(empty) 이진 트리를 생성

    - **라인 6**: 빈 이진 트리이므로 트리의 루트 노드의 참조를 저장하는 변수인 root를 None으로 설정

NOTE: 극소수의 연구용 트리를 제외한 대부분의 트리는 **루트 노드부터 연산이 시작**

- preorder(node) **시간복잡도: O(N)**

```
8      def preorder(self, node):
9          if node != None:
10             print(str(node.get_key()),' ', end='')
11             if node.get_left():
12                 self.preorder(node.get_left())
13             if node.get_right():
14                 self.preorder(node.get_right())
```

  – 이진 트리 T를 전위 순회

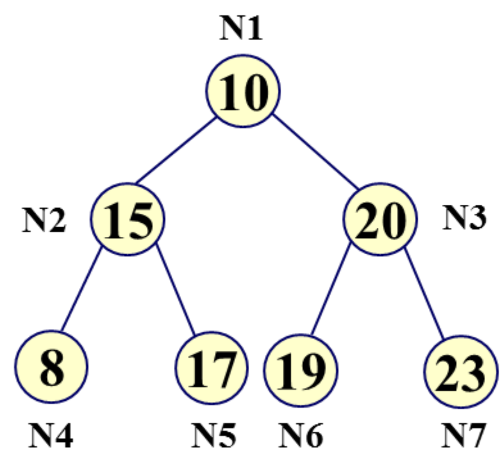
    - **라인 10**: N을 방문(N에 대한 작업을 수행, e.g., N의 key 값을 출력)
    - **라인 11-12**: N의 왼쪽 자식 노드 N_{left}가 None이 아니라면 N_{left}를 루트로 하는 서브 트리 T1을 전위 순회(재귀호출)
    - **라인 13-14**: N의 오른쪽 자식 노드 N_{right}가 None이 아니라면 N_{right}를 루트로 하는 서브 트리 T2를 전위 순회(재귀호출)

이진 트리의 구현(8/13)

❑ 노드들의 참조를 이용하여 연결 시키는 구현 contd.

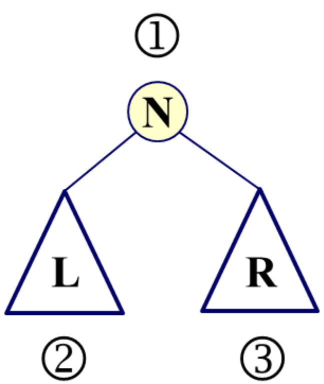
- preorder(node) contd.

```python
    def preorder(self, node):
        if node != None:
            print(str(node.get_key()),' ', end='')
            if node.get_left():
                self.preorder(node.get_left())
            if node.get_right():
                self.preorder(node.get_right())
```



예제 이진 트리

- 이진 트리 T를 전위 순회
  - 예제 이진 트리의 전위 순회 및 방문 순서:
    - » N1(**방문**: 10 출력) → N2(**방문**: 15 출력) → N4(**방문**: 8 출력, N2에게 return None) → N5(**방문**: 17 출력, N2에게 return None) → N2(N1에게 return None) → N3(**방문**: 20 출력) → N6(**방문**: 19 출력, N3에게 return None) → N7(**방문**: 23 출력, N3에게 return None) → N3(N1에게 return None) → N1(호출자에게 return None)
    - » N1 → N2 → N4 → N5 → N3 → N6 → N7 (10 → 15 → 8 →17 → 20 → 19 → 23)
    - » T에 존재하는 모든 노드를 방문해야 하므로 시간 복잡도는 O(N)



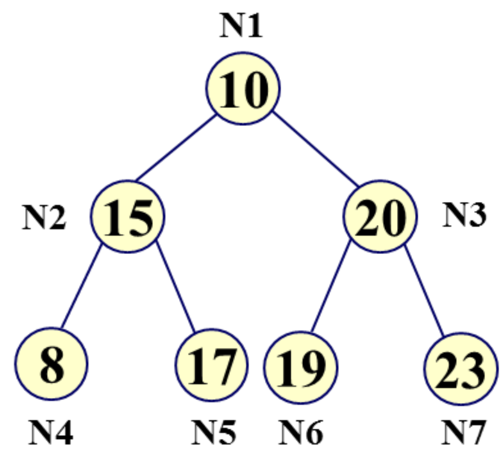
전위순회: NLR

이진 트리의 구현(9/13)

❑ 노드들의 참조를 이용하여 연결 시키는 구현 contd.

- inorder(node) **시간복잡도: O(N)**

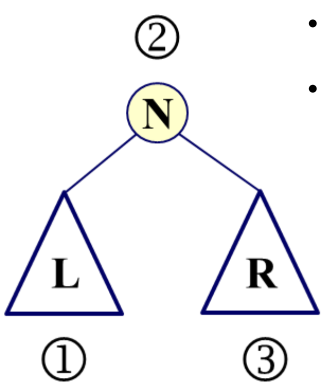
```
16      def inorder(self, node):
17          if node != None:
18              if node.get_left():
19                  self.inorder(node.get_left())
20              print(str(node.get_key()),' ', end='')
21              if node.get_right():
22                  self.inorder(node.get_right())
```



예제 이진 트리

– 이진 트리 T를 중위 순회

- **라인 18-19**: N의 왼쪽 자식 노드 N_{left}가 None이 아니라면 N_{left}를 루트로 하는 서브 트리 T1을 중위 순회(재귀호출)
- **라인 20:** N을 방문(N에 대한 작업을 수행, e.g., N의 key 값을 출력)
- **라인 21-22**: N의 오른쪽 자식 노드 N_{right}가 None이 아니라면 N_{right}를 루트로 하는 서브 트리 T2를 중위 순회(재귀호출)
- 예제 이진 트리의 중위 순회 및 방문 순서
  - » N1 → N2 → N4(**방문**: 8 출력, N2에게 return None) → N2(**방문**: 15 출력) → N5(**방문**: 17 출력, N2에게 return None) → N2(N1에게 return None) → N1(**방문**: 10 출력) → N3 → N6(**방문**: 19 출력, N3에게 return None) → N3(**방문**: 20 출력) → N7(**방문**: 23 출력) → N3(N1에게 return None) → N1(호출자에게 return None)
  - » N4 → N2 → N5 → N1 → N6 → N3 → N7 (8 → 15 → 17 → 10 → 19 → 20 → 23)
  - » T에 존재하는 모든 노드를 방문해야 하므로 시간 복잡도는 O(N)



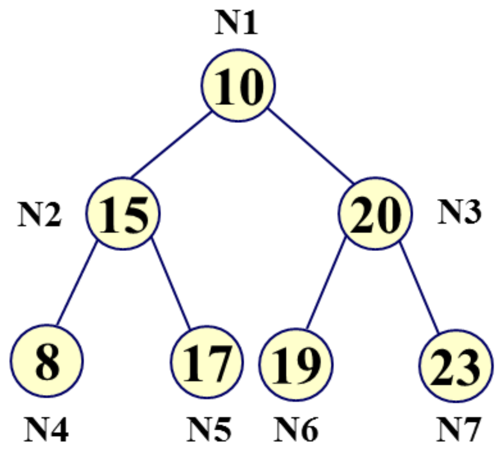
중위순회: LNR

이진 트리의 구현(10/13)

❑ 노드들의 참조를 이용하여 연결 시키는 구현 contd.

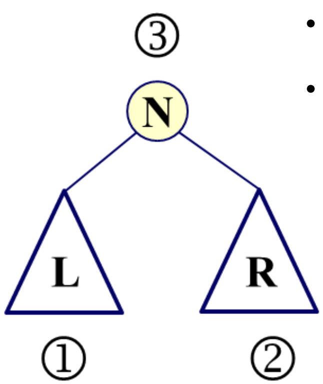
- postorder(node) **시간복잡도: O(N)**

```
24     def postorder(self, node):
25         if node != None:
26             if node.get_left():
27                 self.postorder(node.get_left())
28             if node.get_right():
29                 self.postorder(node.get_right())
30             print(str(node.get_key()),' ', end='')
```



예제 이진 트리

- 이진 트리 T를 후위 순회
  - **라인 26-27**: N의 왼쪽 자식 노드 N_{left}가 None이 아니라면 N_{left}를 루트로 하는 서브 트리 T1을 중위 순회(재귀호출)
  - **라인 28-29**: N의 오른쪽 자식 노드 N_{right}가 None이 아니라면 N_{right}를 루트로 하는 서브 트리 T2를 중위 순회(재귀호출)
  - **라인 30**: N을 방문(N에 대한 작업을 수행, e.g., N의 key 값을 출력)
  - 예제 이진 트리의 중위 순회 및 방문 순서
    - » N1 → N2 → N4(**방문**: 8 출력, N2에게 return None) → N5(**방문**: 17 출력, N2에게 return None) → N2(**방문**: 15 출력, N1에게 return None) → N3 → N6(**방문**: 19 출력, N3에게 return None) → N7(**방문**: 23 출력, N3에게 return None) → N3(**방문**: 20 출력, N1에게 return None) → N1(**방문**: 10 출력, 호출자에게 return None)
    - » N4 → N5 → N2→ N6 → N7 → N3 → N1 (8 → 17 → 15 → 19 → 23 → 20 → 10)
    - » T에 존재하는 모든 노드를 방문해야 하므로 시간 복잡도는 O(N)



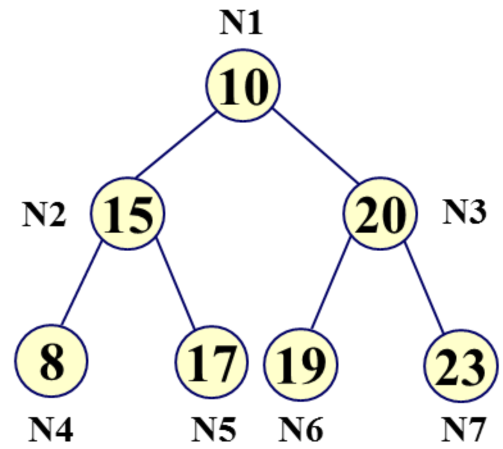
후위순회: LRN

이진 트리의 구현(11/13)

❑ 노드들의 참조를 이용하여 연결 시키는 구현 contd.

- levelorder(node) **시간복잡도: O(N)**

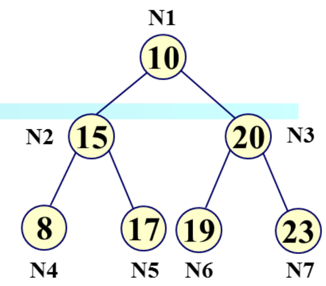
```
32     def levelorder(self, root):
33         q = Queue()
34         q.enqueue(root)
35         while not q.is_empty():
36             node = q.dequeue()
37             print(str(node.get_key()), ' ', end='')
38             if node.get_left():
39                 q.enqueue(node.get_left())
40             if node.get_right():
41                 q.enqueue(node.get_right())
```



예제 이진 트리

  – 이진 트리 T를 레벨 순회
    - **라인 23-34**: empty 큐를 생성 후 T의 루트 노드를 enqueue
    - **라인 35**: 큐가 empty가 될 때까지 while-루프를 실행
    - **라인 36-37**: 큐에서 노드 N을 dequeue 후 N을 출력
    - **라인 38-39**: N의 왼쪽 자식 노드 N_{left}가 None이 아니라면 큐에 N_{left}를enqueue
    - **라인 40-41**: N의 오른쪽 자식 노드 N_{right}가 None이 아니라면 큐에 N_{right}를 enqueue
    - T에 존재하는 모든 노드를 방문해야 하므로 시간 복잡도는 O(N)

이진 트리의 구현(12/13)



예제 이진 트리

N1

OUT IN

N1을 enqueue

N1을 dequeue 후 방문(10 출력)

N2 N3

N1의 왼쪽 자식 노드와 오른쪽 자식 노드를 차례로 enqueue

N3

N2를 dequeue 후 방문(15 출력)

N3 N4 N5

N2의 왼쪽 자식 노드와 오른쪽 자식 노드를 차례로 enqueue

계속

N4 N5

N3을 dequeue 후 방문(20출력)

N4 N5 N6 N7

N3의 왼쪽 자식 노드와 오른쪽 자식 노드를 차례로 enqueue

N5 N6 N7

N4를 dequeue 후 방문(8출력)

N6 N7

N5를 dequeue 후 방문(17출력)

N7

N6를 dequeue 후 방문(19출력)

N7를 dequeue 후 방문(23출력)

큐가 empty이므로 while-루프 탈출

예제 이진 트리의 레벨 순회 및 방문 순서

이진 트리의 구현(13/13)

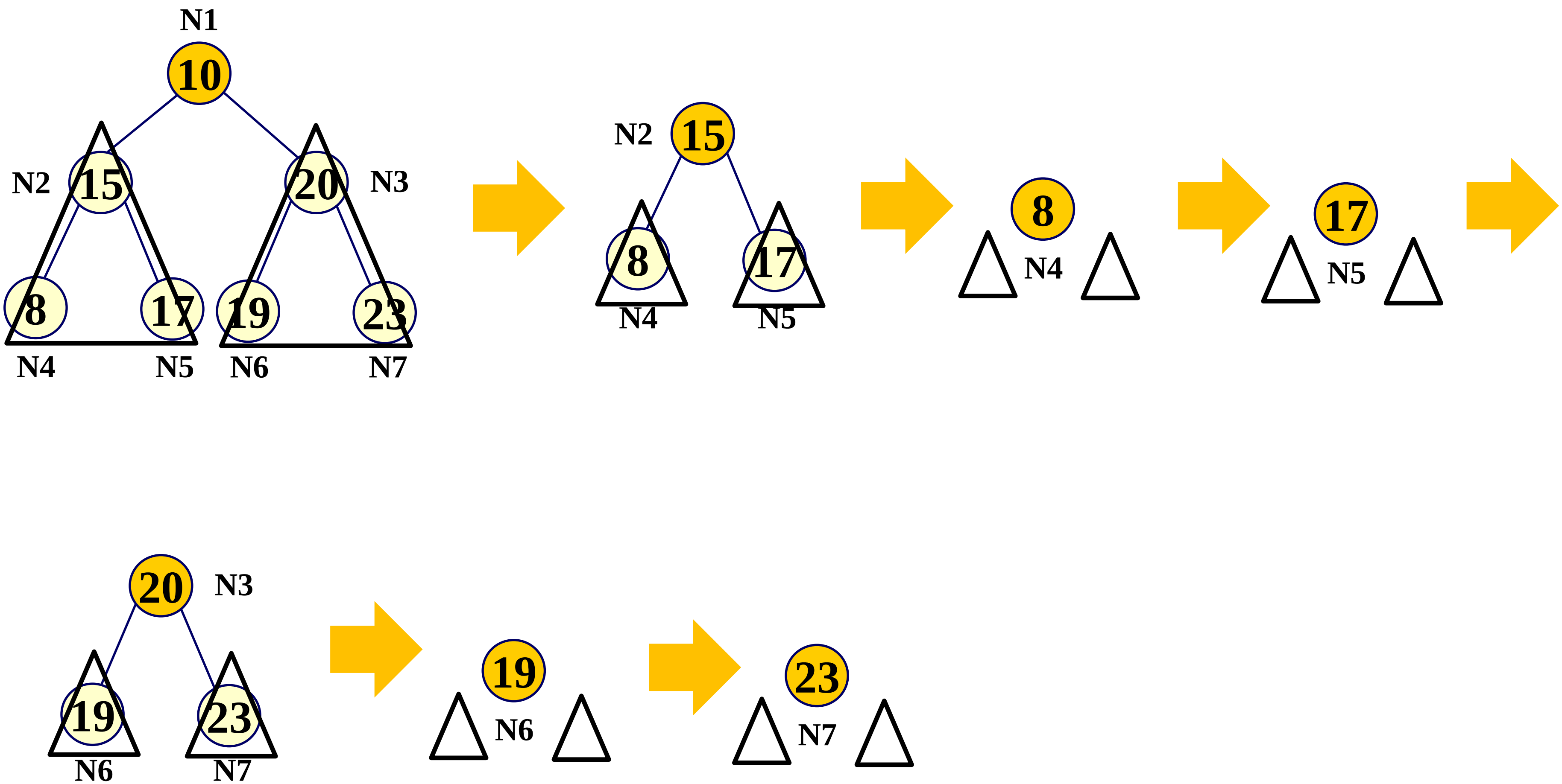
❑ 노드들의 참조를 이용하여 연결 시키는 구현 contd.

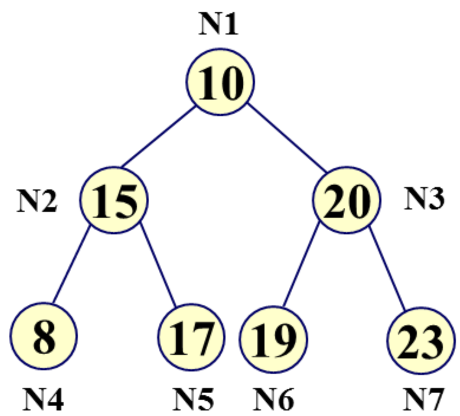
```
if __name__ == '__main__':
    t = BinaryTree()
    n1 = Node(100)
    n2 = Node(200)
    n3 = Node(300)
    n4 = Node(400)
    n5 = Node(500)
    n6 = Node(600)
    n7 = Node(700)
    n8 = Node(800)
    n1.set_left(n2)
    n1.set_right(n3)
    n2.set_left(n4)
    n2.set_right(n5)
    n3.set_left(n6)
    n3.set_right(n7)
    n4.set_left(n8)
    t.root = n1
    print('전위순회:\t', end='')
    t.preorder(t.root)
    print('\n중위순회:\t', end='')
    t.inorder(t.root)
    print('\n후위순회:\t', end='')
    t.postorder(t.root)
    print('\n레벨순회:\t', end='')
    t.levelorder(t.root)
```

일련의 이진 트리 연산과 출력

전위순회:	100	200	400	800	500	300	600	700
중위순회:	800	400	200	500	100	600	300	700
후위순회:	800	400	500	200	600	700	300	100
레벨순회:	100	200	300	400	500	600	700	800

결과


N1
10
N2
15
20
N3
8
17
19
23
N4
N5
N6
N7
N2
15
8
17
N4
N5
8
N4
17
N5
20
N3
19
23
N6
N7
19
N6
23
N7



N1

10

```
def preorder(self, node):
    print(str(node.get_key()),' ', end='')
    if node.get_left():
        self.preorder(node.get_left())
    if node.get_right():
        self.preorder(node.get_right())
```

N2

15

```
def preorder(self, node):
    print(str(node.get_key()),' ', end='')
    if node.get_left():
        self.preorder(node.get_left())
    if node.get_right():
        self.preorder(node.get_right())
```

N4

8

```
def preorder(self, node):
    print(str(node.get_key()),' ', end='')
    if node.get_left():
        self.preorder(node.get_left())
    if node.get_right():
        self.preorder(node.get_right())
```

N5

17

```
def preorder(self, node):
    print(str(node.get_key()),' ', end='')
    if node.get_left():
        self.preorder(node.get_left())
    if node.get_right():
        self.preorder(node.get_right())
```

N3

20

```
def preorder(self, node):
    print(str(node.get_key()),' ', end='')
    if node.get_left():
        self.preorder(node.get_left())
    if node.get_right():
        self.preorder(node.get_right())
```

N6

19

```
def preorder(self, node):
    print(str(node.get_key()),' ', end='')
    if node.get_left():
        self.preorder(node.get_left())
    if node.get_right():
        self.preorder(node.get_right())
```

N7

23

```
def preorder(self, node):
    print(str(node.get_key()),' ', end='')
    if node.get_left():
        self.preorder(node.get_left())
    if node.get_right():
        self.preorder(node.get_right())
```