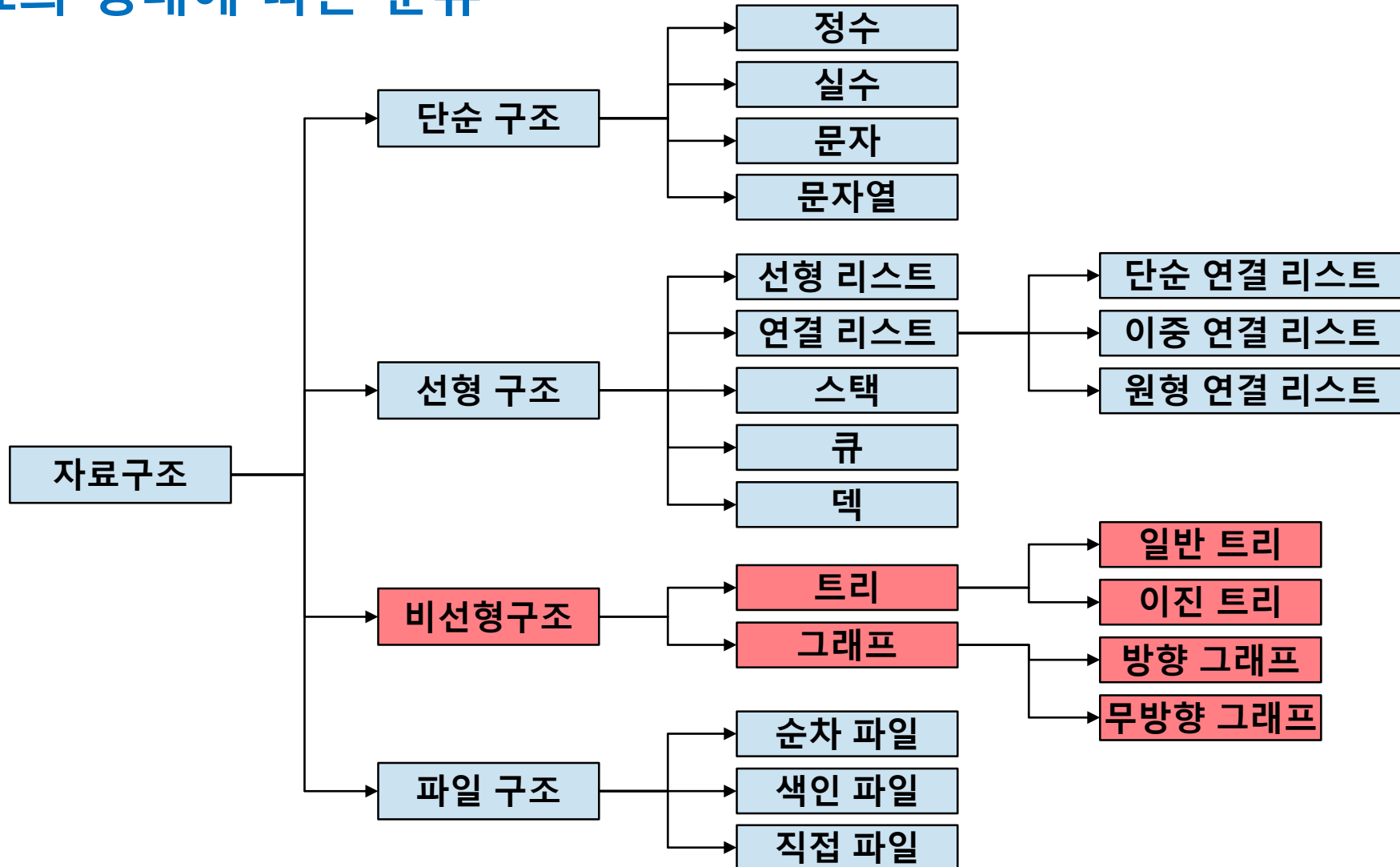


# 데이터구조와 알고리즘 - 트리(Tree) I

남춘성

# Tree

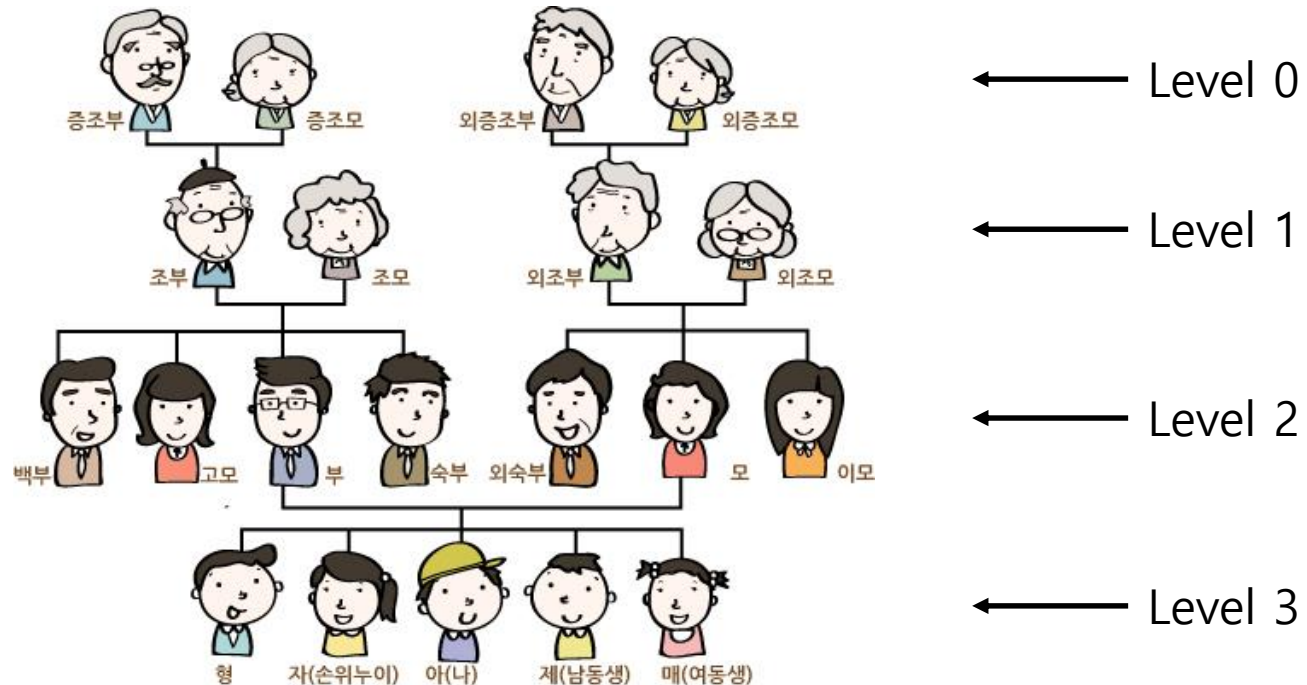
- 자료의 형태에 따른 분류



# Tree

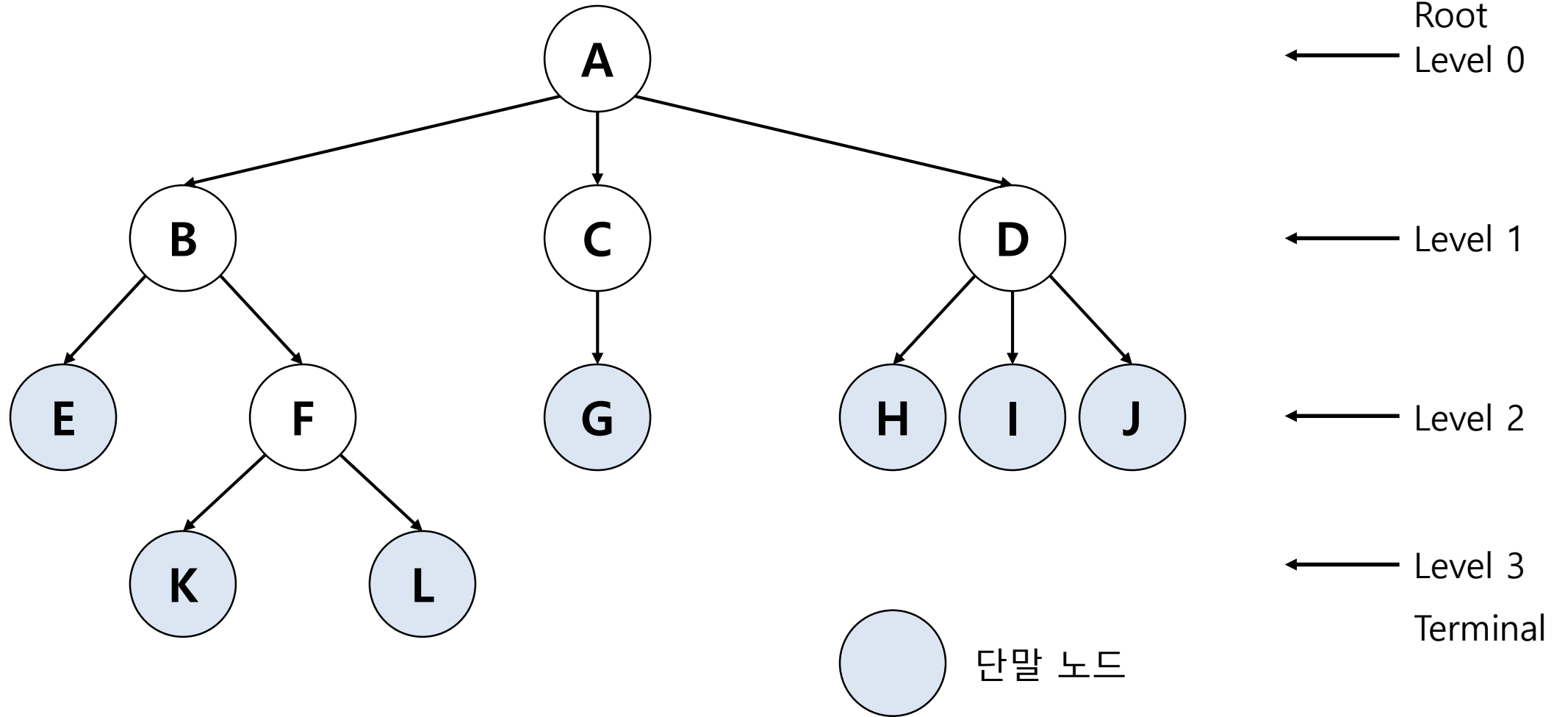
- 트리(tree)

- 원소들 간에 1:n 관계를 가지는 비선형 자료구조
- 원소들 간에 계층관계를 가지는 계층형 자료구조
- 상위 원소에서 하위 원소로 내려가면서 확장되는 트리(나무)모양의 구조
- 트리 자료구조의 예 - 가계도
  - 가계도의 자료 : 가족 구성원
  - 자료를 연결하는 선 : 부모-자식 관계 표현



# Tree

- 트리 A



# Tree

- 트리 A

- **노드(node)**: 트리의 원소
  - 트리 A의 노드: A,B,C,D,E,F,G,H,I,J,K,L
- **루트 노드(root node)**: 트리의 시작 노드
  - 트리 A의 루트노드: A
- **간선(edge)**: 노드를 연결하는 선. 부모 노드와 자식 노드를 연결
- **형제 노드(sibling node)**: 같은 부모 노드의 자식 노드들
  - B,C,D는 형제 노드
- **조상 노드**: 간선을 따라 루트 노드까지 이르는 경로에 있는 모든 노드들
  - K의 조상 노드 : F, B, A
- **서브 트리(subtree)**: 부모 노드와 연결된 간선을 끊었을 때 생성되는 트리
  - 각 노드는 자식 노드의 개수 만큼 서브 트리를 가진다.
- **자손 노드**: 서브 트리에 있는 하위 레벨의 노드들
  - B의 자손 노드: E,F,K,L

# Tree

- 트리 A

- 차수(degree)

- 노드의 차수 : 노드에 연결된 자식 노드의 수.

- A의 차수=3, B의 차수=2, C의 차수=1

- 트리의 차수 : 트리에 있는 노드의 차수 중에서 가장 큰 값

- 트리 A의 차수=3

- 단말 노드(리프 노드) : 차수가 0인 노드. 자식 노드가 없는 노드

- 높이

- 노드의 높이 : 루트에서 노드에 이르는 간선의 수. 노드의 레벨

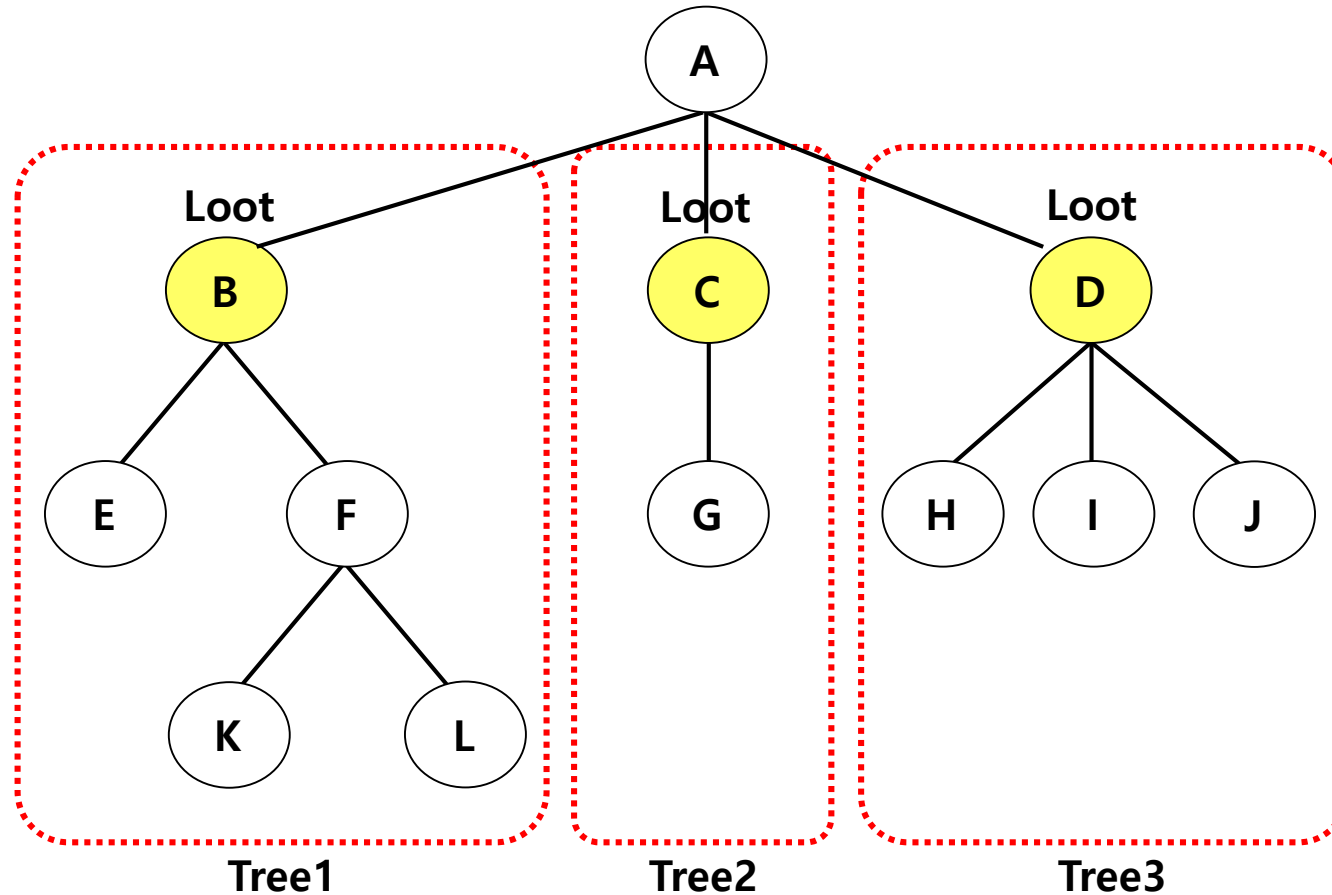
- B의 높이=1, F의 높이=2

- 트리의 높이 : 트리에 있는 노드의 높이 중에서 가장 큰 값. 최대 레벨

- 트리 A의 높이=3

# Tree

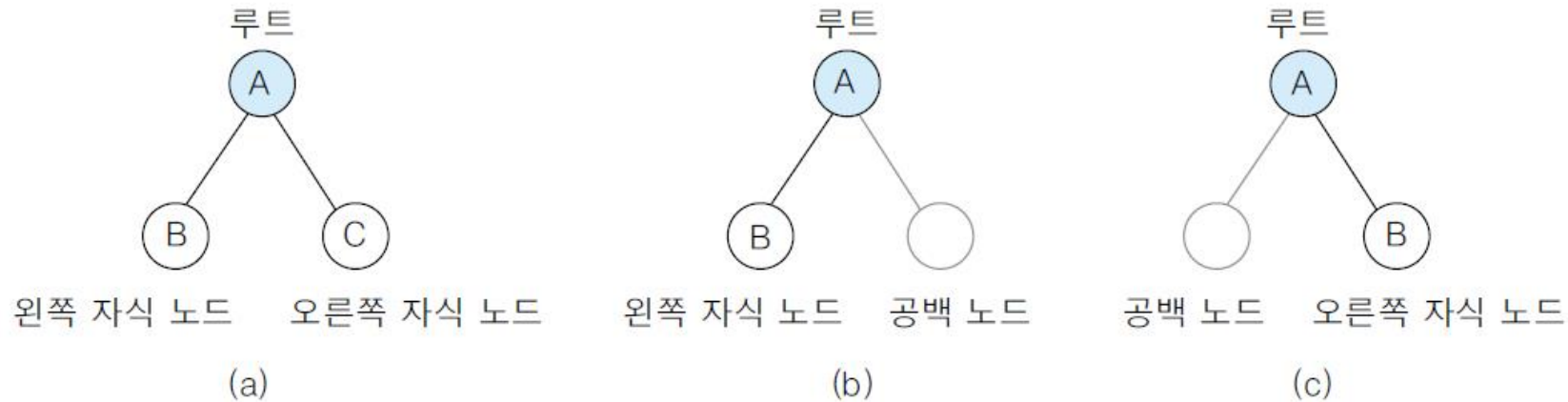
- 포리스트(forest) : 서브트리의 집합
  - 트리A에서 노드 A를 제거하면
    - A의 자식 노드 B, C, D에 대한 서브 트리가 생성
    - 이들의 집합은 포리스트



# Tree

- 이진트리

- 트리의 노드 구조를 일정하게 정의하여 트리의 구현과 연산이 쉽도록 정의한 트리
- 이진 트리의 모든 노드는 **왼쪽 자식 노드**와 **오른쪽 자식 노드** 만을 가진다.
  - 부모 노드와 자식 노드 수와의 관계 → 1:2
  - 공백 노드도 자식 노드로 취급한다.
  - $0 \leq \text{노드의 차수} \leq 2$



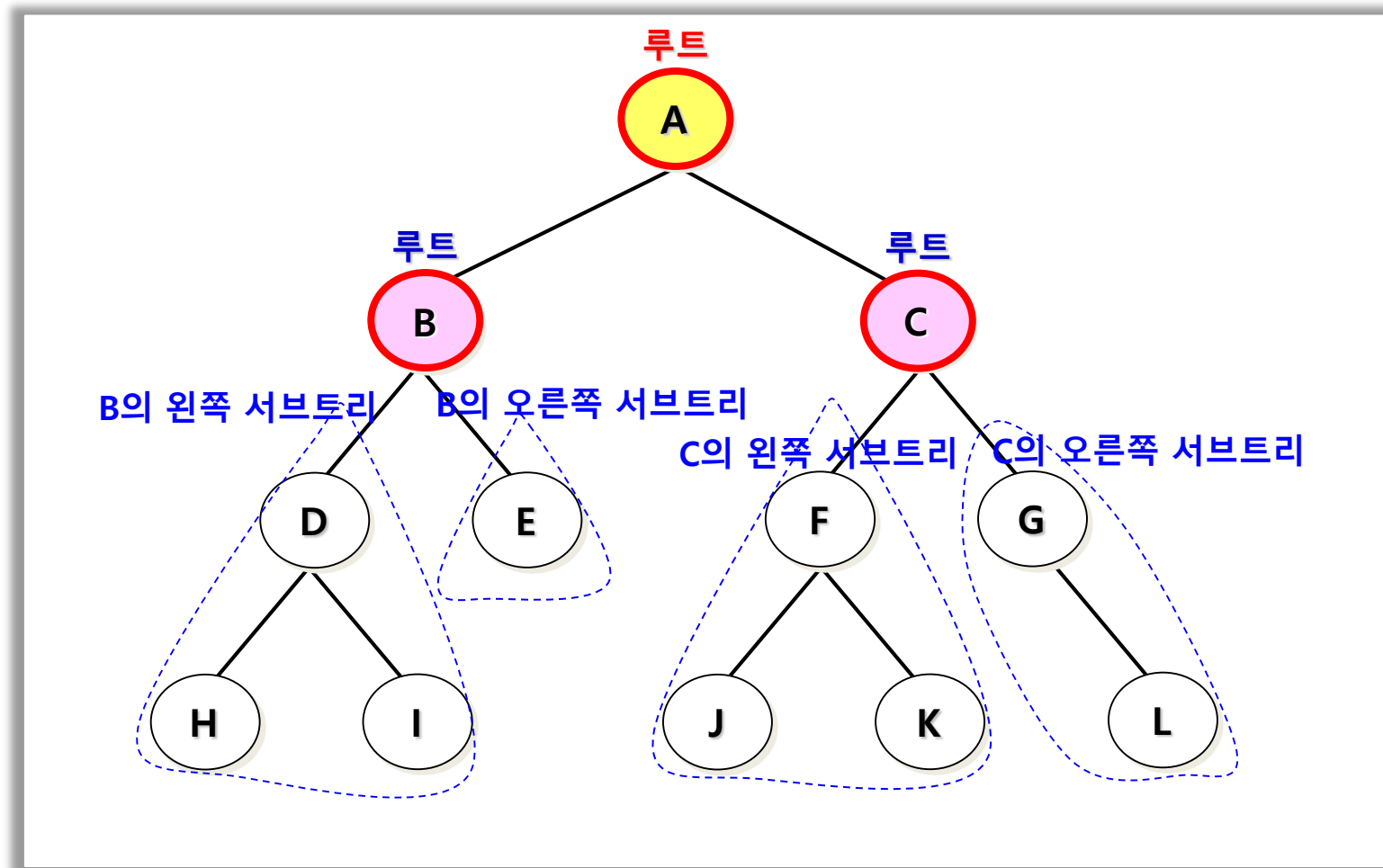
[그림 8-4] 이진 트리의 기본 구조



# Tree

- 이진트리는 순환적 구성

- 노드의 왼쪽 자식 노드를 루트로 하는 왼쪽 서브트리도 이진 트리
- 노드의 오른쪽 자식 노드를 루트로 하는 오른쪽 서브 트리도 이진 트리



# Tree

- 추상 자료형 이진트리

## ADT BinaryTree

데이터 : 공백이거나 루트 노드, 왼쪽 서브 트리, 오른쪽 서브 트리로 구성된  
노드들의 유한 집합

연산 :

$bt, bt1, bt2 \in \text{BinaryTree}; \text{item} \in \text{Element};$

**createBT()** ::= create an empty binary tree;

// 공백 이진 트리를 생성하는 연산

**isEmpty(bt)** ::= if (bt is empty) then return true  
else return false;

// 이진 트리가 공백인지 아닌지를 확인하는 연산

**makeBT(bt1, item, bt2)** ::= return {item을 루트로 하고 bt1을 왼쪽 서브 트리,  
bt2를 오른쪽 서브 트리로 하는 이진 트리}

// 두 개의 이진 서브 트리를 연결하여 하나의 이진 트리를 만드는 연산

**leftSubtree(bt)** ::= if (isEmpty(bt)) then return null  
else return left subtree of bt;

// 이진 트리의 왼쪽 서브 트리를 구하는 연산

**rightSubtree(bt)** ::= if (isEmpty(bt)) then return null  
else return right subtree of bt;

// 이진 트리의 오른쪽 서브 트리를 구하는 연산

**data(bt)** ::= if (isEmpty(bt)) then return null  
else return the item in the root node of bt;

// 이진 트리에서 루트 노드의 데이터(item)를 구하는 연산

**End BinaryTree**

# Tree

- **이진트리의 특성**

**정의1)**  $n$ 개의 노드를 가진 이진 트리는 항상  $(n-1)$ 개의 간선을 가진다.

- 루트를 제외한  $(n-1)$ 개의 노드가 부모 노드와 연결되는 한 개의 간선을 가짐

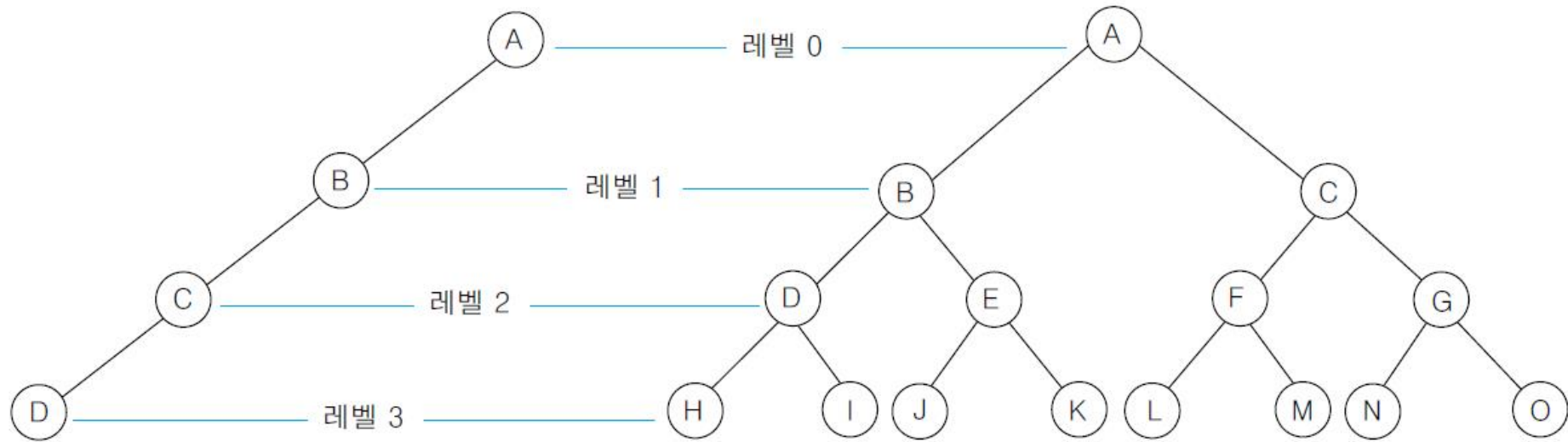
**정의2)** 높이가  $h$ 인 이진 트리가 가질 수 있는 노드의 최소 개수는  $(h+1)$ 개가 되며, 최대 개수는  $(2^{h+1}-1)$ 개가 된다.

- 이진 트리의 높이가  $h$ 가 되려면 한 레벨에 최소한 한 개의 노드가 있어야 하므로 높이가  $h$ 인 이진 트리의 최소 노드의 개수는  $(h+1)$ 개
- 하나의 노드는 최대 2개의 자식 노드를 가질 수 있으므로 레벨  $i$ 에서의 노드의 최대 개수는  $2^i$ 개 이므로 높이가  $h$ 인 이진 트리 전체의 노드 개수는
- $\sum 2^i = 2^{h+1} - 1$  개

# Tree

- 이진트리의 특성

- 높이가 3이면서 최소의 노드를 갖는 이진트리와 최대의 노드를 갖는 이진트리



(a) 최소의 노드를 갖는 이진 트리

(b) 최대의 노드를 갖는 이진 트리

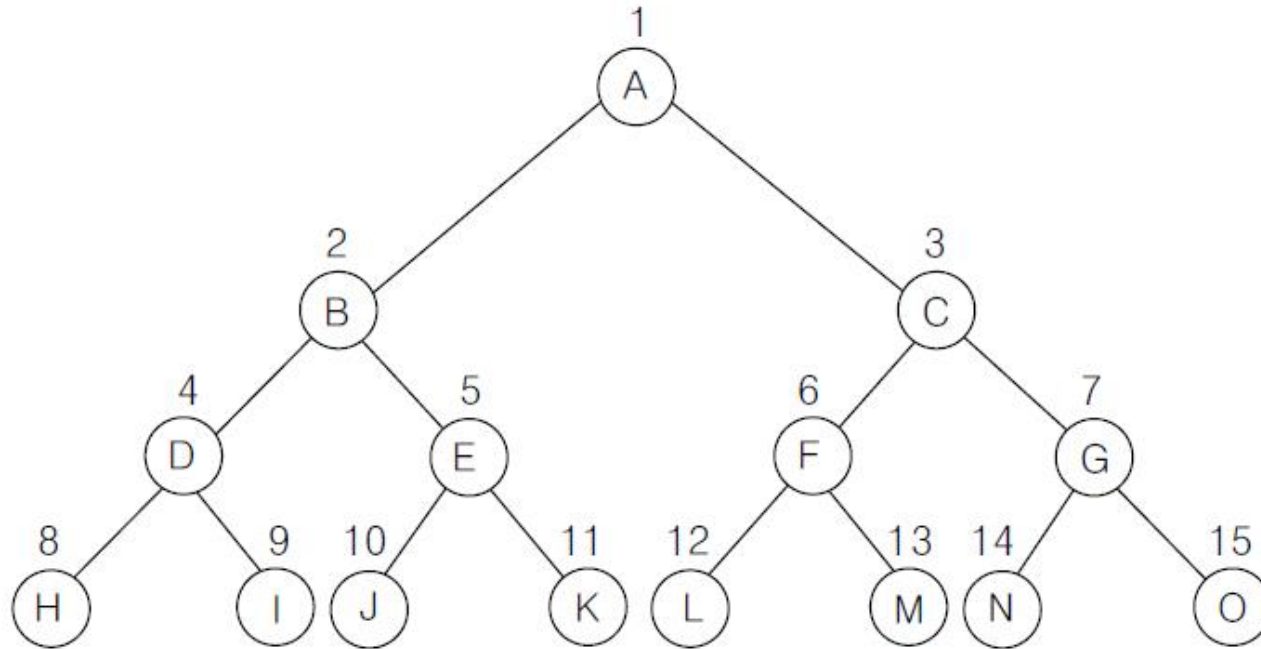
[그림 8-6] 높이가 3이면서 최소의 노드를 갖는 이진 트리와 최대의 노드를 갖는 이진 트리

# Tree

- 이진 트리의 종류

- 포화 이진 트리(Full Binary Tree)

- 모든 레벨에 노드가 포화상태로 차 있는 이진 트리
- 높이가  $h$ 일 때, 최대의 노드 개수인  $(2^{h+1}-1)$  의 노드를 가진 이진 트리
- 루트를 1번으로 하여  $2^{h+1}-1$ 까지 정해진 위치에 대한 노드 번호를 가짐

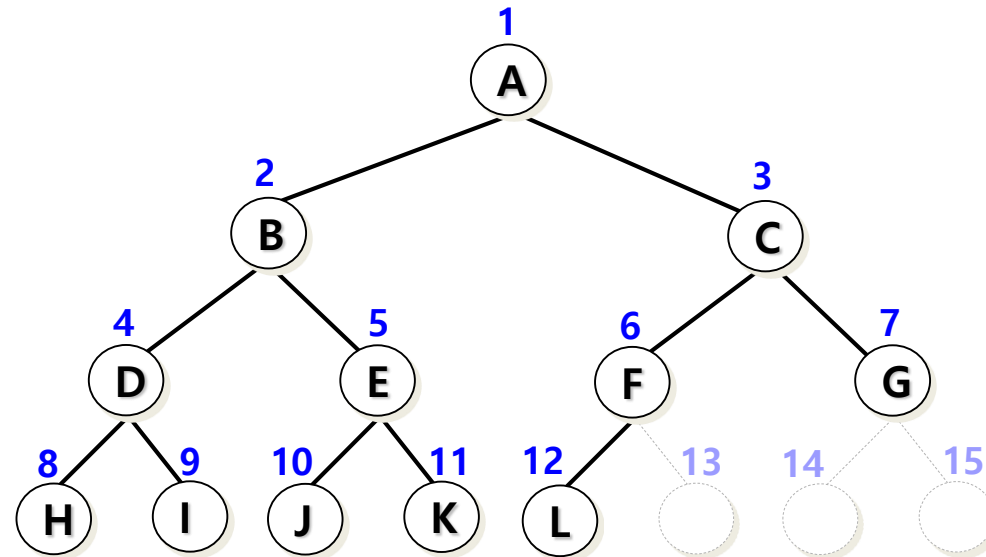


[그림 8-7] 높이가 3인 포화 이진 트리

# Tree

- **완전 이진 트리(Complete Binary Tree)**

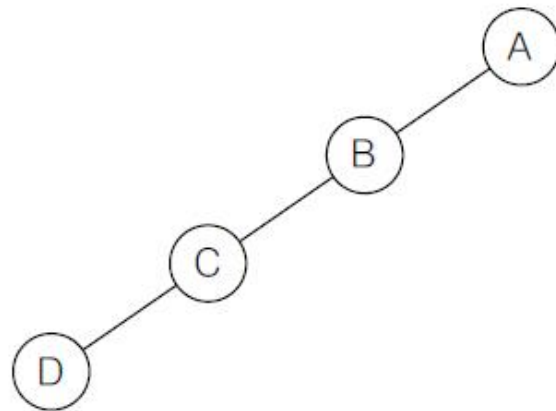
- 높이가  $h$ 이고 노드 수가  $n$ 개일 때 (단,  $h+1 \leq n < 2^{h+1}-1$ ), 포화 이진 트리의 노드 번호 1번부터  $n$ 번까지 빈 자리가 없는 이진 트리(왼쪽부터 차례차곡 채워진 상태) 예) 노드가 12개인 완전 이진 트리



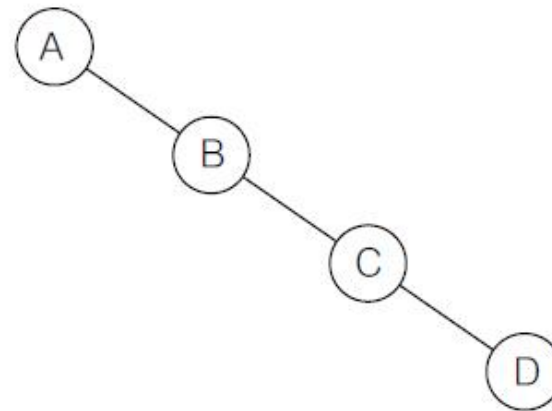
# Tree

- **편향 이진 트리(Skewed Binary Tree)**

- 높이  $h$ 에 대한 최소 개수의 노드를 가지면서 한쪽 방향의 자식 노드만을 가진 이진 트리
- 왼쪽 편향 이진 트리
  - 모든 노드가 왼쪽 자식 노드만을 가진 편향 이진 트리
- 오른쪽 편향 이진 트리
  - 모든 노드가 오른쪽 자식 노드만을 가진 편향 이진 트리



(a) 왼쪽 편향 이진 트리



(b) 오른쪽 편향 이진 트리

[그림 8-9] 높이가 3인 편향 이진 트리

# Tree

- 순차 자료구조를 이용한 이진트리의 구현

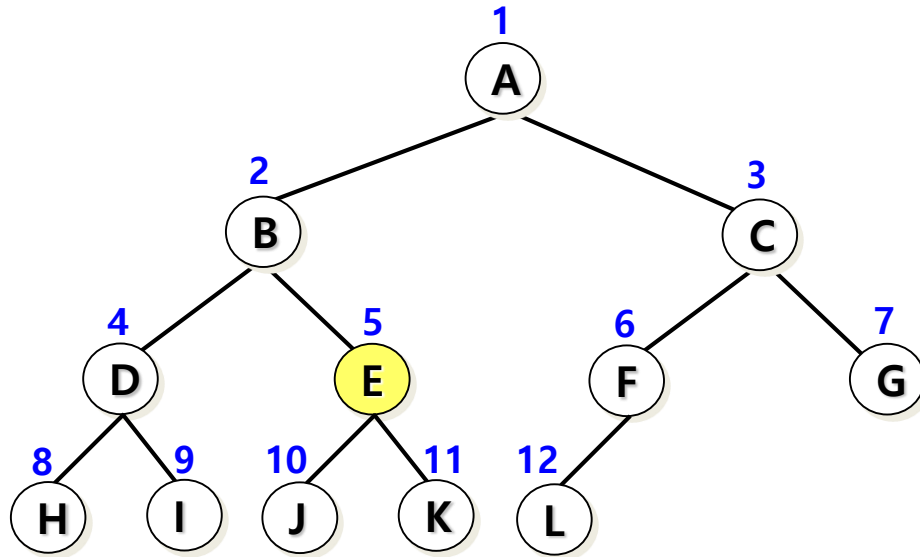
- 1차원 배열의 순차 자료구조 사용

- 높이가  $h$ 인 포화 이진 트리의 노드번호를 배열의 인덱스로 사용
    - 인덱스 0번 : 실제로 사용하지 않고 비워둠.
    - 인덱스 1번 : 루트 저장



# Tree

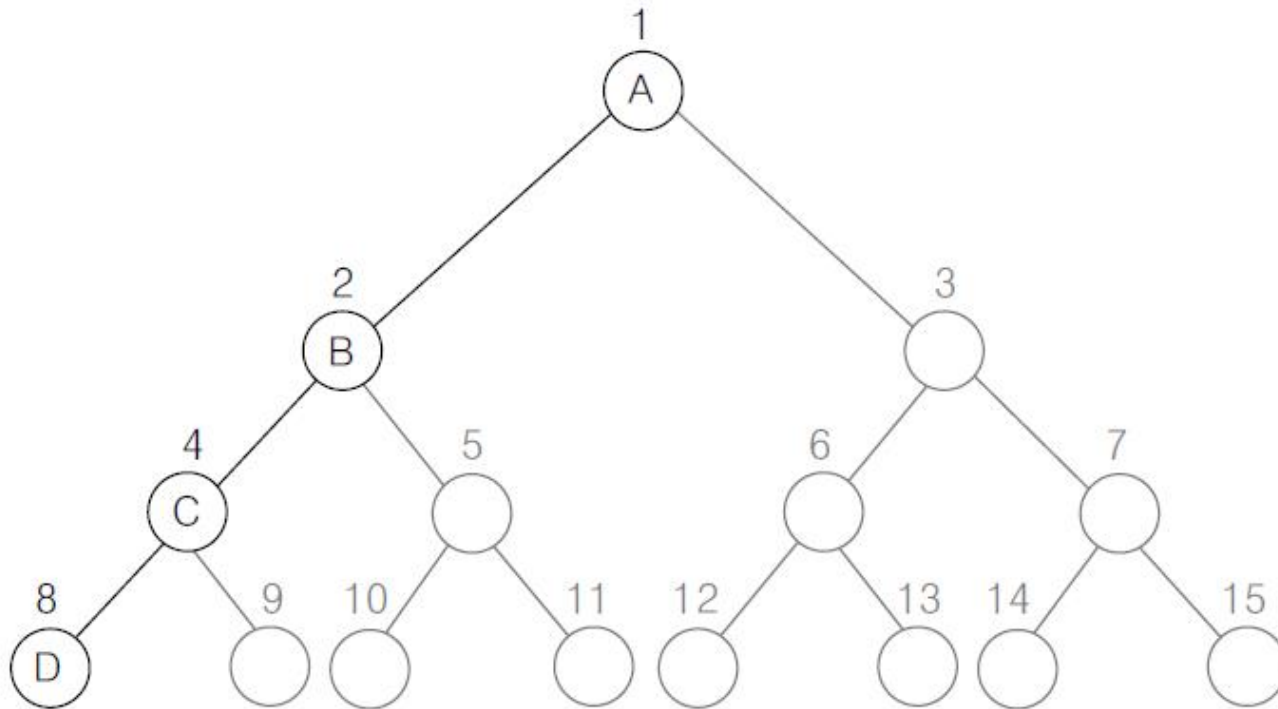
- 완전 이진 트리의 1차원 배열 표현



[0]	
[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I
[10]	J
[11]	K
[12]	L

# Tree

- 왼쪽 편향 이진 트리의 1차원 배열 표현



[0]	
[1]	A
[2]	B
[3]	
[4]	C
[5]	
[6]	
[7]	
[8]	D

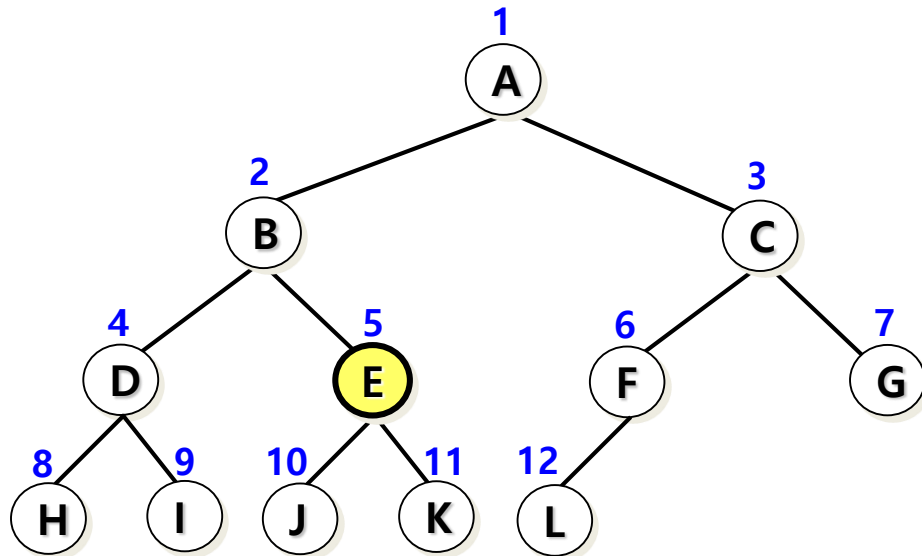
[그림 8-11] 편향 이진 트리의 배열 표현

# Tree

## 이진 트리의 1차원 배열에서의 인덱스 관계

노드	인덱스	성립 조건
노드 $i$ 의 부모 노드	$\lfloor i/2 \rfloor$	$i > 1$
노드 $i$ 의 왼쪽 자식 노드	$2 \times i$	$(2 \times i) \leq n$
노드 $i$ 의 오른쪽 자식 노드	$(2 \times i) + 1$	$(2 \times i + 1) \leq n$
루트 노드	1	$n > 0$

전체 노드의 수 =  $n$   
 $i$  = 노드의 수



# Tree

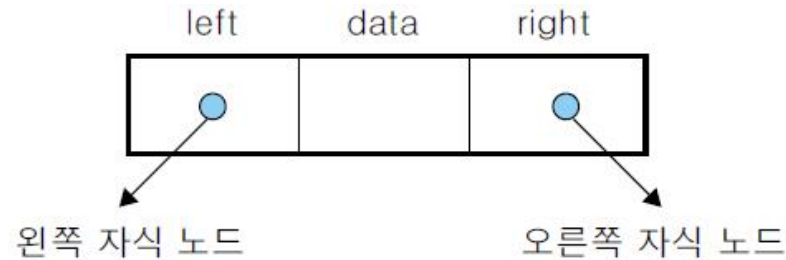
- **이진 트리의 순차 자료구조 표현의 단점**

- 편향 이진 트리의 경우에 사용하지 않는 배열 원소에 대한 메모리 공간 낭비 발생
- 트리의 원소 삽입/삭제에 대한 배열의 크기 변경 어려움

# Tree

- 연결 자료구조를 이용한 이진트리의 구현

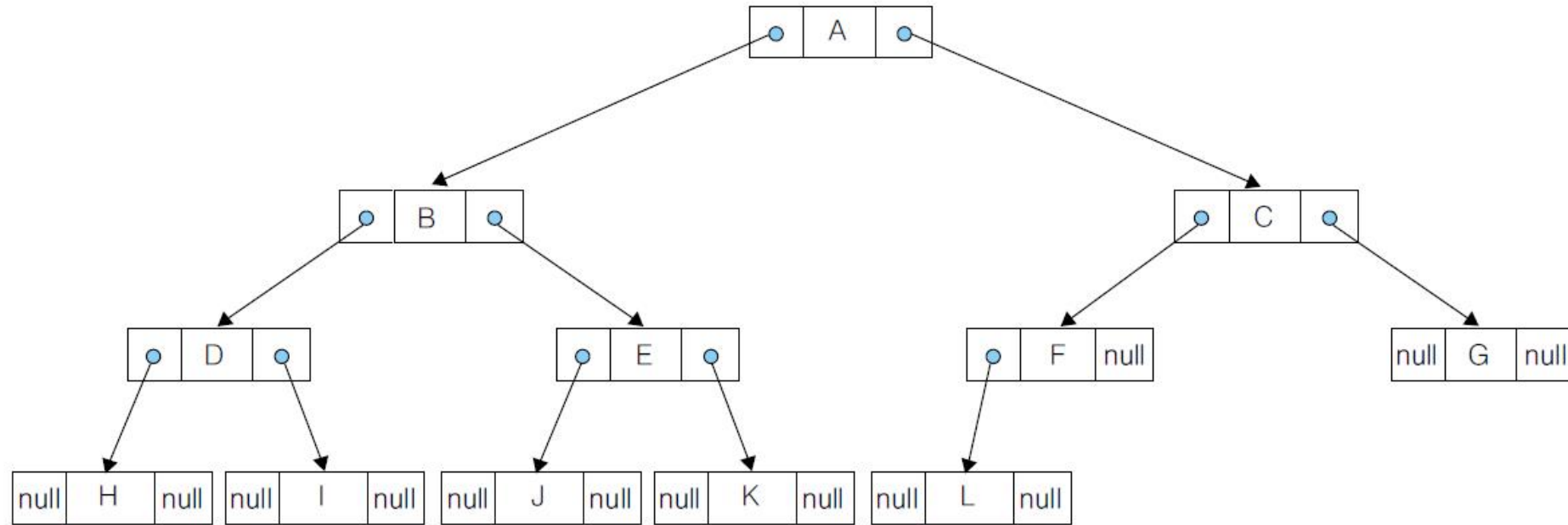
- 단순 연결 리스트를 사용하여 구현
- 이진 트리의 모든 노드는 최대 2개의 자식 노드를 가지므로 일정한 구조의 단순 연결 리스트 노드를 사용하여 구현



[그림 8-12] 이진 트리의 노드 구조

# Tree

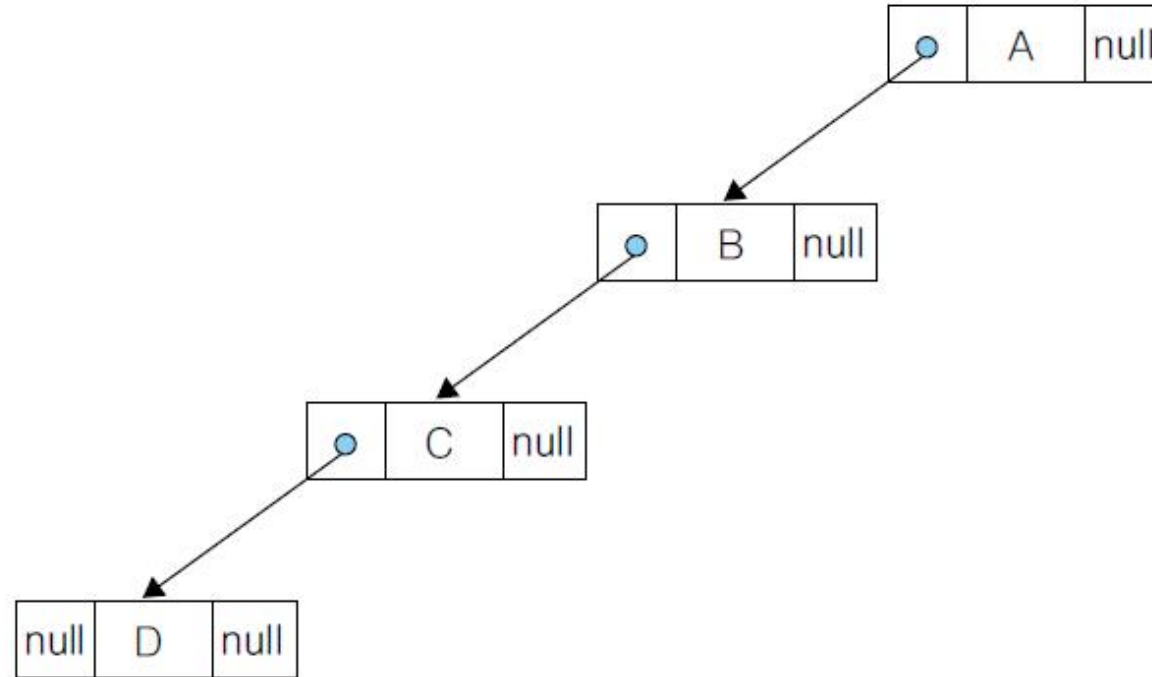
- 완전 이진 트리의 단순 연결 리스트 표현



(a) 완전 이진 트리

# Tree

- 왼쪽 편향 이진 트리의 단순 연결 리스트 표현



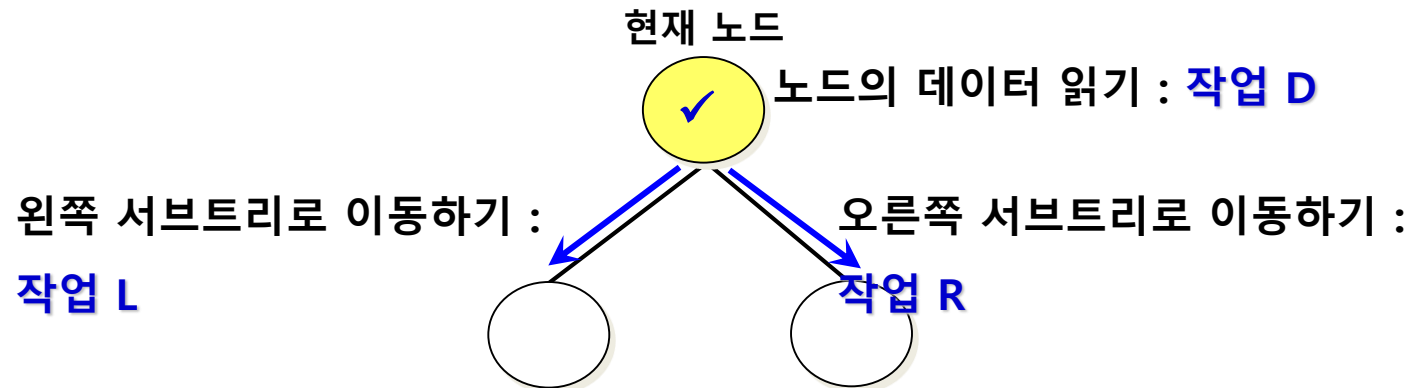
(b) 편향 이진 트리

[그림 8-13] 이진 트리의 연결 자료구조 표현

# Tree

- 이진 트리의 순회(traversal)

- 계층적 구조로 저장되어있는 트리의 모든 노드를 방문하여 데이터를 처리하는 연산
- 순회를 위해 수행할 수 있는 작업 정의
  - (1) 현재 노드를 방문하여 데이터를 읽는 작업 **D**
  - (2) 현재 노드의 왼쪽 서브트리로 이동하는 작업 **L**
  - (3) 현재 노드의 오른쪽 서브트리로 이동하는 작업 **R**





# Tree

- 이진 트리가 순환적으로 정의되어 구성되어있으므로, 순회작업도 서브트리에 대해서 순환적으로 반복하여 완성
- 왼쪽 서브트리에 대한 순회를 오른쪽 서브트리 보다 먼저 수행
- 순회의 종류
  - 전위 순회
  - 중위 순회
  - 후위 순회

# Tree

- 전위 순회(preorder traversal)

- 수행 방법

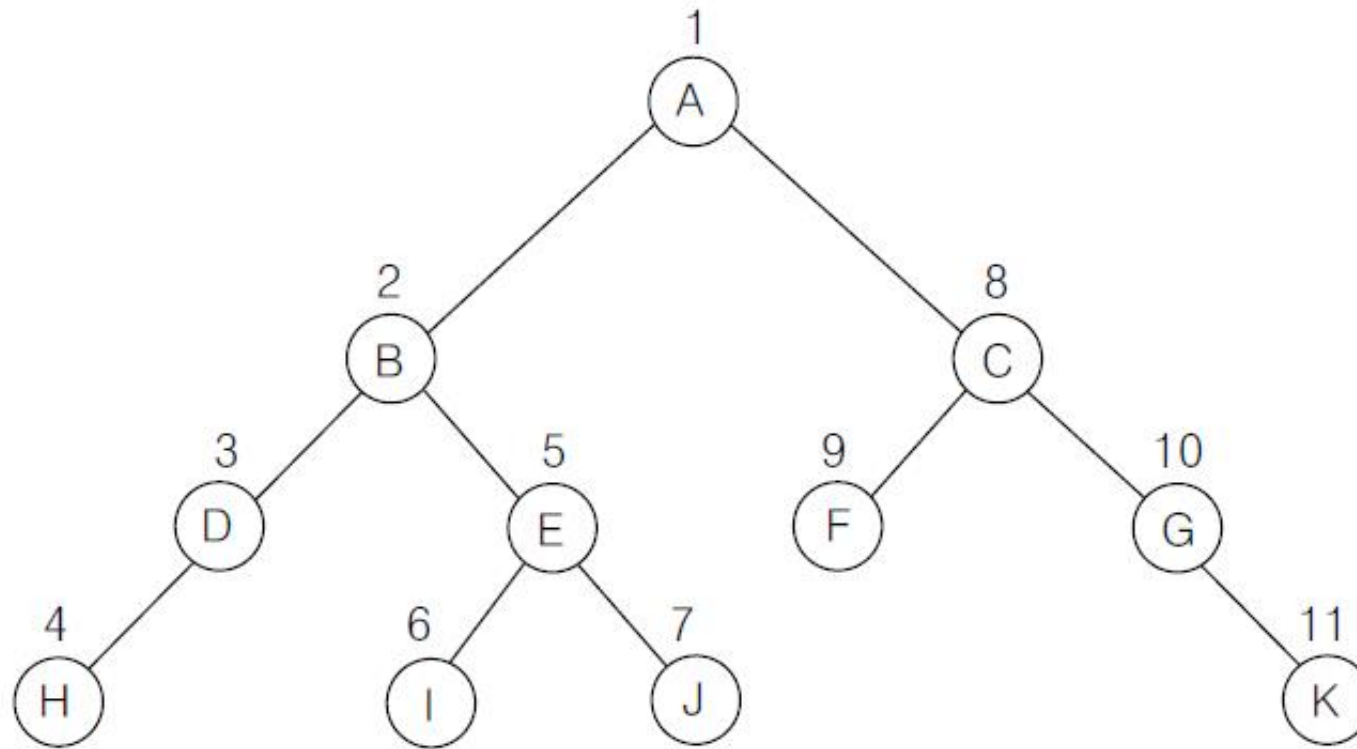
- ① 현재 노드  $n$ 을 방문하여 처리한다. : **D**
- ② 현재 노드  $n$ 의 왼쪽 서브트리로 이동한다. : **L**
- ③ 현재 노드  $n$ 의 오른쪽 서브트리로 이동한다. : **R**

- 전위 순회 알고리즘

```
preorder(T)
    if(T≠null) then {
        visit T.data;
        preorder(T.left);
        preorder(T.right);
    }
end preorder
```

# Tree

- 전위 순회 (D-L-R)의 예
- 이진트리의 전위 순회 경로: A-B-D-H-E-I-J-C-F-G-K



[그림 8-14] 이진 트리의 전위 순회 경로 : A-B-D-H-E-I-J-C-F-G-K

# Tree

- 전위 순회 과정 -> A-B-D-H-E-I-J-C-F-G-K

- ① 노드 A( $\odot$ LR) - 루트A에서 전위 순회를 시작하여 현재 노드 A의 데이터를 읽고,  
노드 A( $\odot$ L $\odot$ R)  $\rightarrow$  노드 B - 왼쪽 서브트리인 노드 B로 이동한다.
- ② 노드 A( $\odot$ L $\odot$ R)  $\rightarrow$  노드 B( $\odot$ LR) - 현재 노드 B의 데이터를 읽고,  
노드 A( $\odot$ L $\odot$ R)  $\rightarrow$  노드 B( $\odot$ L $\odot$ R)  $\rightarrow$  노드 D - 왼쪽 서브트리인 노드 D로 이동한다.
- ③ 노드 A( $\odot$ L $\odot$ R)  $\rightarrow$  노드 B( $\odot$ L $\odot$ R)  $\rightarrow$  노드 D( $\odot$ LR) - 현재 노드 D의 데이터를 읽는다.
- ④ 노드 A( $\odot$ L $\odot$ R)  $\rightarrow$  노드 B( $\odot$ L $\odot$ R)  $\rightarrow$  노드 D( $\odot$ L $\odot$ R)  $\rightarrow$  노드 H - 현재 노드 D의 왼쪽 단말노드 H의 데이터를 읽고,  
노드 A( $\odot$ L $\odot$ R)  $\rightarrow$  노드 B( $\odot$ L $\odot$ R)  $\rightarrow$  노드 D( $\odot$ L $\odot$  $\otimes$ )  $\rightarrow$  공백 노드 - 노드 D의 오른쪽 노드인 공백 노드를 읽는 것으로 노드 D에 대한 DLR 순회가 끝난다.  
노드 A( $\odot$ L $\odot$ R)  $\rightarrow$  노드 B( $\odot$ L $\odot$ R)  $\leftarrow$  ~~노드 D( $\odot$ L $\odot$  $\otimes$ )~~ - 노드 D의 순회가 끝났으므로 이전 경로인 노드 B로 돌아간다.  
노드 A( $\odot$ L $\odot$ R)  $\rightarrow$  노드 B( $\odot$ L $\odot$  $\otimes$ )  $\rightarrow$  노드 E - 현재 노드 B의 오른쪽 서브트리인 노드 E로 이동한다.

# Tree

- 전위 순회 과정 -> A-B-D-H-E-I-J-C-F-G-K

⑤ 노드 A(DLR) → 노드 B(DLR) → 노드 E(DLR) - 현재 노드 E의 데이터를 읽는다.

⑥ 노드 A(DLR) → 노드 B(DLR) → 노드 E(DLR) → 노드 I - 노드 E의 왼쪽 단말 노드 I의 데이터를 읽는다.

⑦ 노드 A(DLR) → 노드 B(DLR) → 노드 E(DLR) → 노드 J - 노드 E의 오른쪽 단말 노드 J의 데이터를 읽는다.

노드 A(DLR) → 노드 B(DLR) ← ~~노드 E(DLR)~~ - 노드 E에 대한 순회가 끝났으므로 노드 E의 이전 경로인 노드 B로 돌아간다.

노드 A(DLR) ← ~~노드 B(DLR)~~ - 이로써 현재 노드 B에서의 DLR 순회가 끝났으므로 다시 이전 노드 A로 돌아간다.

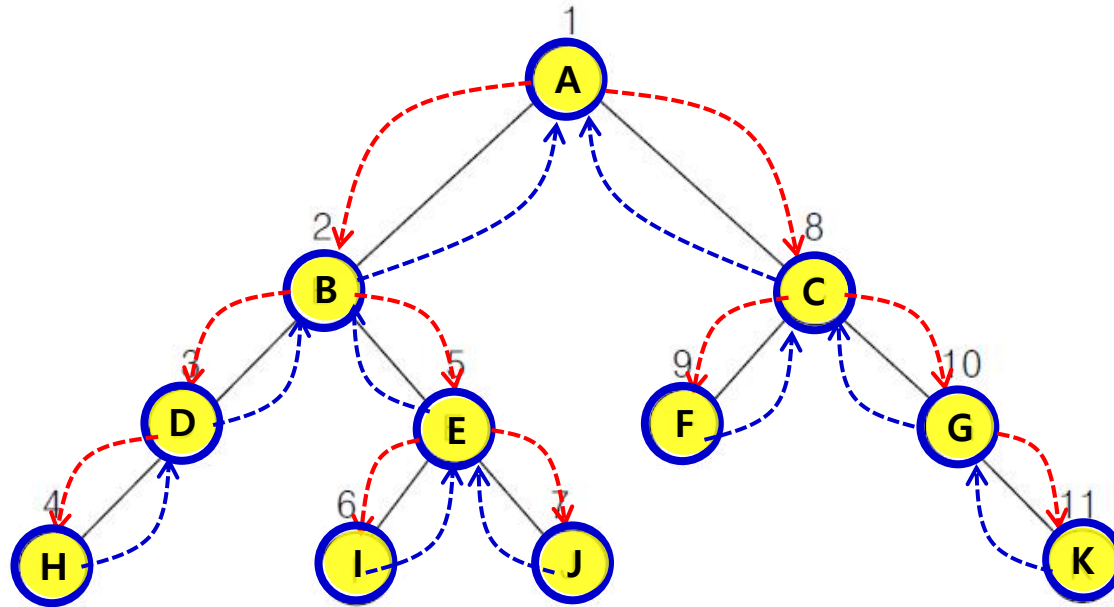
# Tree

- 전위 순회 과정 -> A-B-D-H-E-I-J-C-F-G-K

- ⑧  $\text{노드A}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 C}$  - 현재 노드 A의 오른쪽 서브트리인 노드 C로 이동하여  
 $\text{노드A}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 C}(\textcircled{D}\textcircled{L}\textcircled{R})$  - 현재 노드 C의 데이터를 읽는다.
- ⑨  $\text{노드A}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 C}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 F}$  - 현재 노드 C의 왼쪽 단말 노드 F로 이동하여 데이터를 읽고  
 $\text{노드A}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 C}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 G}$  - 오른쪽 서브트리인 노드 G로 이동한다.
- ⑩  $\text{노드A}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 C}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 G}(\textcircled{D}\textcircled{L}\textcircled{R})$  - 현재 노드 G의 데이터를 읽는다.
- ⑪  $\text{노드A}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 C}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 G}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{공백 노드}$  - 노드 G의 왼쪽 노드인 공백 노드를 읽고,  
 $\text{노드A}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 C}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 G}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 K}$  - 노드 G의 오른쪽 단말 노드 K의 데이터를 읽는다.  
 $\text{노드A}(\textcircled{D}\textcircled{L}\textcircled{R}) \rightarrow \text{노드 C}(\textcircled{D}\textcircled{L}\textcircled{R}) \leftarrow \text{노드 G}(\textcircled{D}\textcircled{L}\textcircled{R})$  - 이로써 노드 G에서의 DLR 순회가 끝났으므로 이전 노드 C로 돌아간다.  
 $\text{노드A}(\textcircled{D}\textcircled{L}\textcircled{R}) \leftarrow \text{노드 C}(\textcircled{D}\textcircled{L}\textcircled{R})$  - 현재 노드 C에서의 DLR 순회 역시 끝났으므로 다시 이전 노드 A로 돌아간다.  
 $\text{노드A}(\textcircled{D}\textcircled{L}\textcircled{R})$  - 이로써 루트 노드 A에 대한 DLR 순회가 끝났으므로 트리 전체에 대한 전위 순회가 완성되었다.

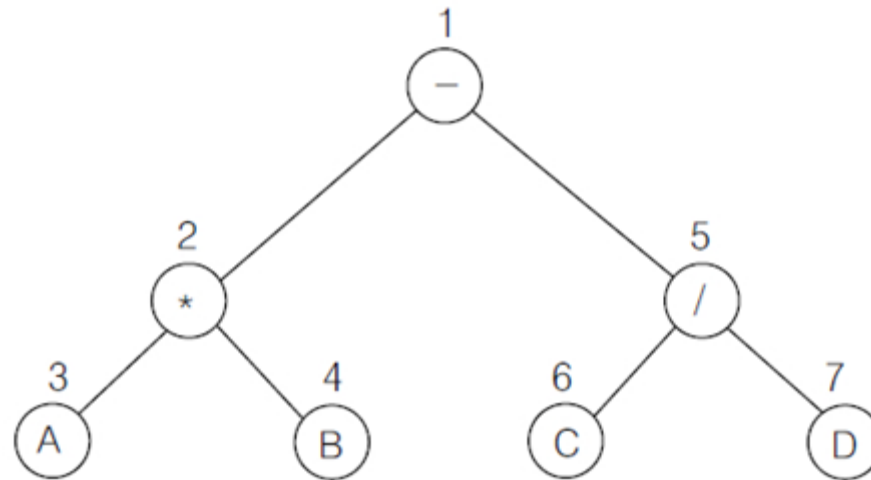
# Tree

- 전위 순회 과정 -> A-B-D-H-E-I-J-C-F-G-K



# Tree

- 수식 이진 트리에 대한 전위 순회
  - 수식을 이진 트리로 구성한 수식 이진 트리를 전위 순회하면, 수식에 대한 전위 표기식을 구할 수 있다.
  - 수식 이진 트리의 전위 순회 경로 >> **-\*AB/CD**





# Tree

- 중위 순회(inorder traversal)

- 수행 방법

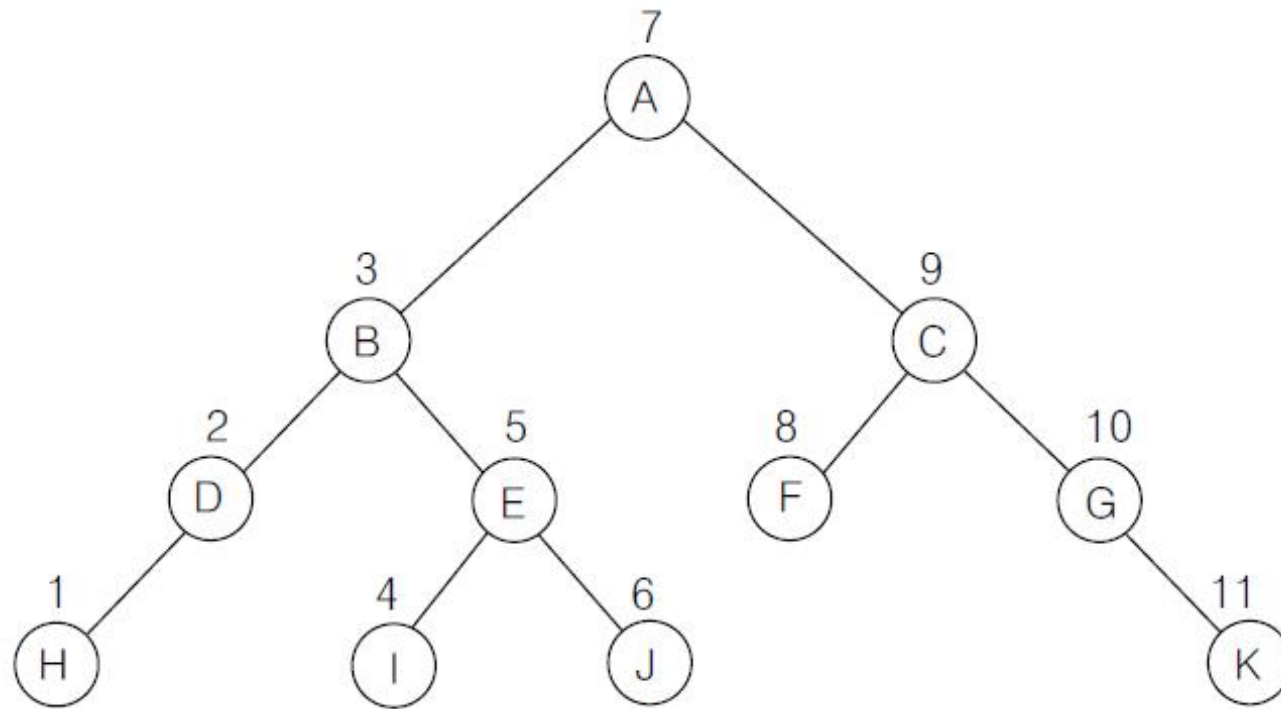
- ① 현재 노드  $n$ 의 왼쪽 서브트리로 이동한다. : **L**
- ② 현재 노드  $n$ 을 방문하여 처리한다. : **D**
- ③ 현재 노드  $n$ 의 오른쪽 서브트리로 이동한다. : **R**

- 중위 순회 알고리즘

```
inorder(T)
    if(T≠null) then {
        inorder(T.left);
        visit T.data;
        inorder(T.right);
    }
end inorder
```

# Tree

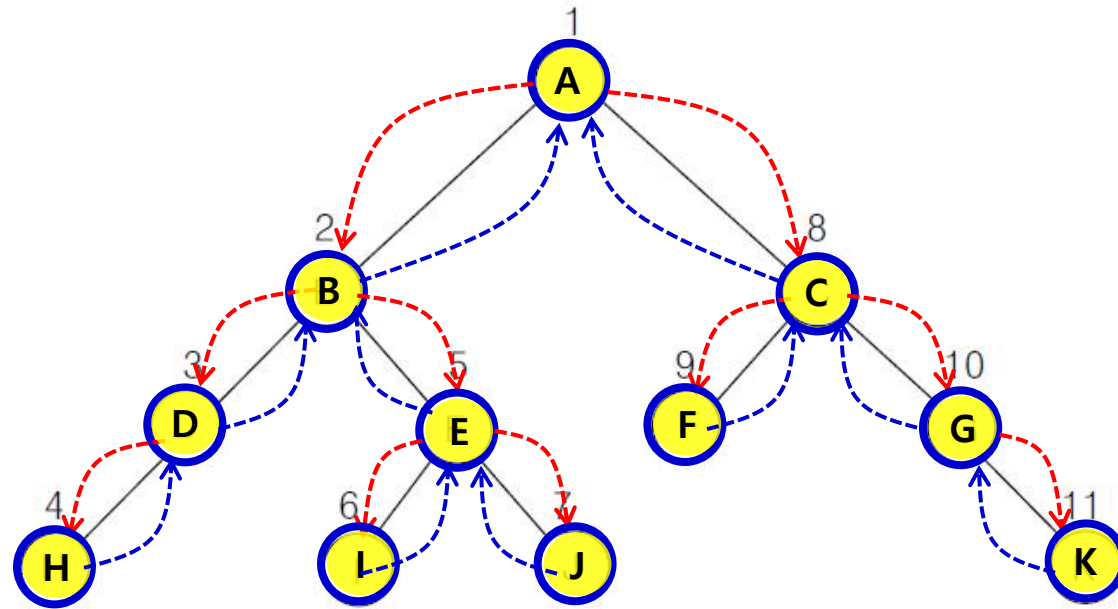
- 중위 순회 (L-D-R)의 예
- 이진트리의 중위 순회 경로: H-D-B-I-E-J-A-F-C-G-K



[그림 8-16] 이진 트리의 중위 순회 경로: H-D-B-I-E-J-A-F-C-G-K

# Tree

- 중위 순회 과정 -> H-D-B-I-E-J-A-F-C-G-K



① 노드 A(ⓁDR) → 노드 B - 루트 A에서 중위 순회를 시작하여 노드 A의 왼쪽 서브트리 B로 이동한다.

노드 A(ⓁDR) → 노드 B(ⓁDR) → 노드 D - 현재 노드 B의 왼쪽 서브트리 D로 이동한다.

노드 A(ⓁDR) → 노드 B(ⓁDR) → 노드 D(ⓁDR) → 노드 H - 현재 노드 D의 왼쪽 단말 노드 H의 데이터를 읽는다.

# Tree

- 중위 순회 과정 -> H-D-B-I-E-J-A-F-C-G-K

② 노드 A(LDR) → 노드 B(LDR) → 노드 D(LDR) - 현재 노드 D의 데이터를 읽고,  
노드 A(LDR) → 노드 B(LDR) → 노드 D(LDR) → 공백 노드 - 노드 D의 오른쪽 단말 노드  
인 공백 노드를 읽는다.  
노드 A(LDR) → 노드 B(LDR) ← 노드 D(LDR) - 노드 D에서의 LDR 순회가 끝났으므로 이  
전 경로인 노드 B로 돌아간다.

③ 노드 A(LDR) → 노드 B(LDR) - 현재 노드 B의 데이터를 읽고,  
노드 A(LDR) → 노드 B(LDR) → 노드 E - 오른쪽 서브트리 E로 이동한다.

④ 노드 A(LDR) → 노드 B(LDR) → 노드 E(LDR) → 노드 I - 현재 노드 E의 왼쪽 단말 노드 I  
의 데이터를 읽는다.  
⑤ 노드 A(LDR) → 노드 B(LDR) → 노드 E(LDR) - 현재 노드 E의 데이터를 읽는다.

# Tree

- 중위 순회 과정 -> H-D-B-I-E-J-A-F-C-G-K

⑥ 노드  $A(LDR) \rightarrow$  노드  $B(LDR) \rightarrow$  노드  $E(LDR) \rightarrow$  노드 J - 현재 노드 E의 오른쪽 단말 노드 J의 데이터를 읽고,

노드  $A(LDR) \rightarrow$  노드  $B(LDR) \leftarrow$  ~~노드  $E(LDR)$~~  - 노드 E에서의 LDR 순회가 끝났으므로 이전 경로인 노드 B로 돌아간다.

노드  $A(LDR) \leftarrow$  ~~노드  $B(LDR)$~~  - 이로써 현재 노드 B에서의 LDR 순회가 끝났으므로 다시 이전 경로인 노드 A로 돌아간다.

⑦ 노드  $A(LDR) \leftarrow$  현재 노드 A의 데이터를 읽고,

노드  $A(LDR) \rightarrow$  노드 C - 현재 노드 A의 오른쪽 서브트리 C로 이동한다.

⑧ 노드  $A(LDR) \rightarrow$  노드  $C(LDR) \rightarrow$  노드 F - 현재 노드 C의 왼쪽 단말 노드 F의 데이터를 읽는다.

⑨ 노드  $A(LDR) \rightarrow$  노드  $C(LDR) \leftarrow$  현재 노드 C의 데이터를 읽고,

노드  $A(LDR) \rightarrow$  노드  $C(LDR) \rightarrow$  노드 G - 노드 C의 오른쪽 서브트리 G로 이동한다.

# Tree

- 중위 순회 과정 -> H-D-B-I-E-J-A-F-C-G-K

⑩ 노드  $A(LDR) \rightarrow$  노드  $C(LDR) \rightarrow$  노드  $G(LDR) \rightarrow$  공백 노드 - 현재 노드 G의 왼쪽 단말 노드인 공백 노드를 읽고,  
노드  $A(LDR) \rightarrow$  노드  $C(LDR) \rightarrow$  노드  $G(LDR)$  - 노드 G의 데이터를 읽는다.

⑪ 노드  $A(LDR) \rightarrow$  노드  $C(LDR) \rightarrow$  노드  $G(LDR) \rightarrow$  노드 K - 현재 노드 G의 오른쪽 단말 노드 K의 데이터를 읽는다.

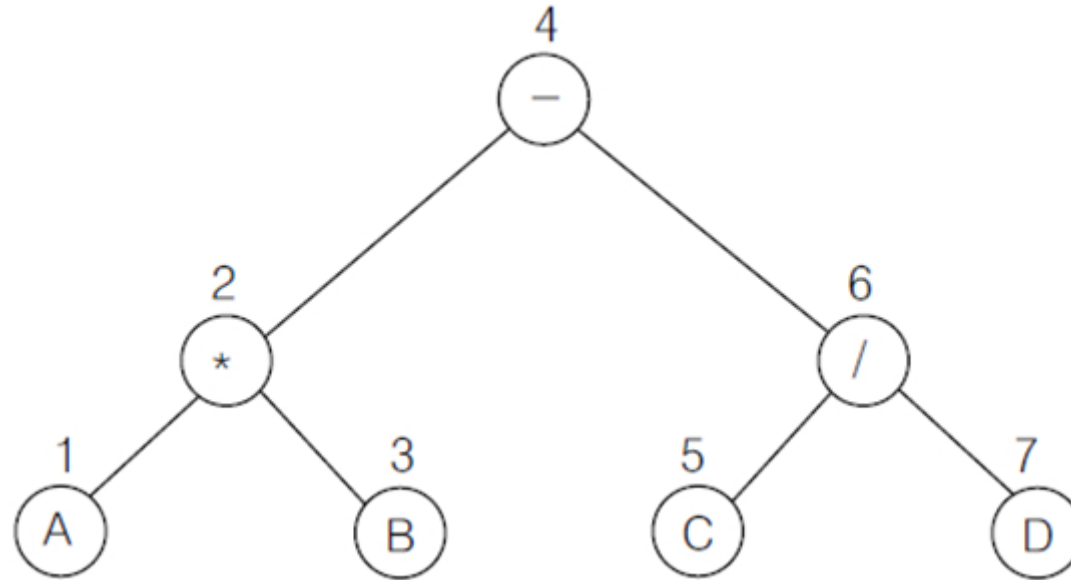
노드  $A(LDR) \rightarrow$  노드  $C(LDR) \leftarrow$  ~~노드  $G(LDR)$~~  - 노드 G에서의 LDR 순회가 끝났으므로 이전 노드 C로 돌아간다.

노드  $A(LDR) \leftarrow$  ~~노드  $C(LDR)$~~  - 현재 노드 C에서의 LDR 순회 역시 끝났으므로 다시 이전 노드 A로 돌아간다.

~~노드  $A(LDR)$~~  - 이로써 루트 노드 A에 대한 LDR 순회가 모두 끝났으므로 트리 전체에 대한 중위 순회가 완성되었다.

# Tree

- 수식 이진 트리에 대한 중위 순회
  - 수식 이진 트리를 중위 순회하면, 수식에 대한 중위 표기식을 구할 수 있다.
  - [그림 8-15]의 수식 이진 트리의 중위 순회 경로 >> **A\*B-C/D**



# Tree

- 후위 순회(postorder traversal)

- 수행 방법

- ① 현재 노드  $n$ 의 왼쪽 서브트리로 이동한다. : **L**
- ② 현재 노드  $n$ 의 오른쪽 서브트리로 이동한다. : **R**
- ③ 현재 노드  $n$ 을 방문하여 처리한다. : **D**

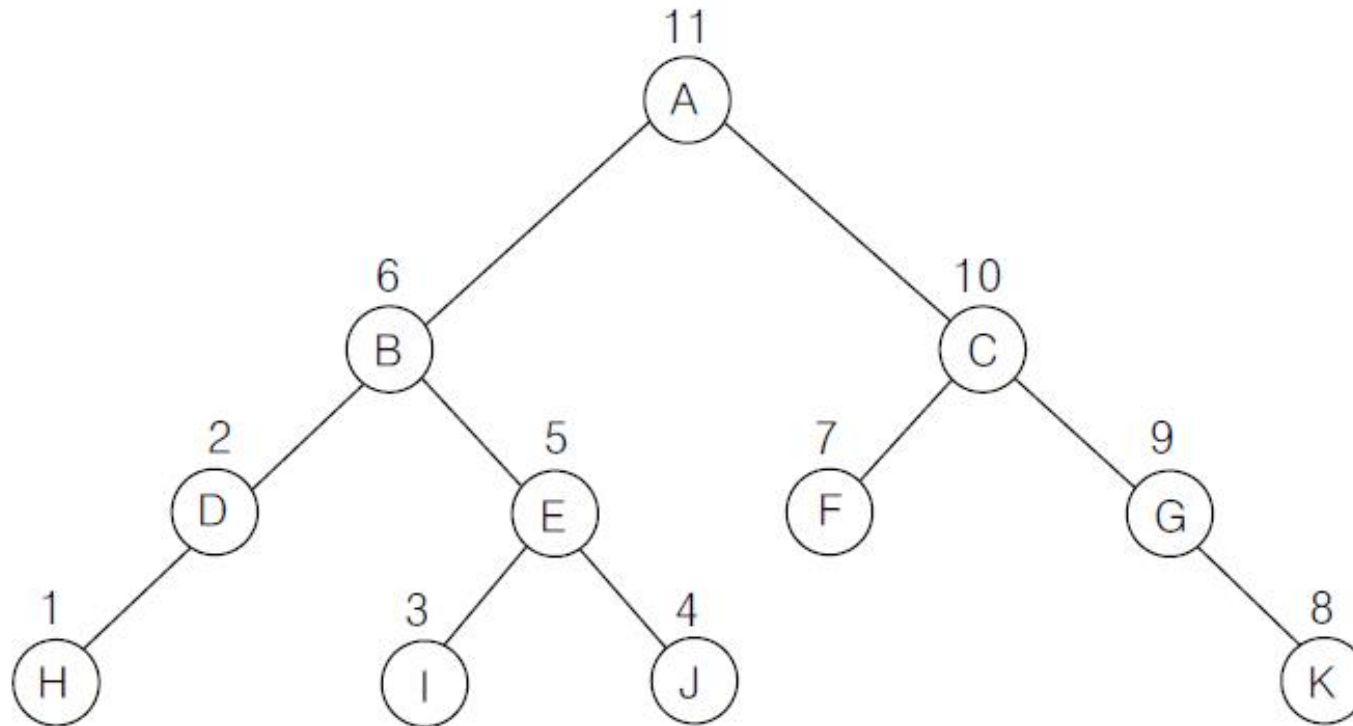
- 후위 순회 알고리즘

```
postorder(T)
    if(T≠null) then {
        postorder(T.left);
        postorder(T.right);
        visit T.data;
    }
end inorder
```



# Tree

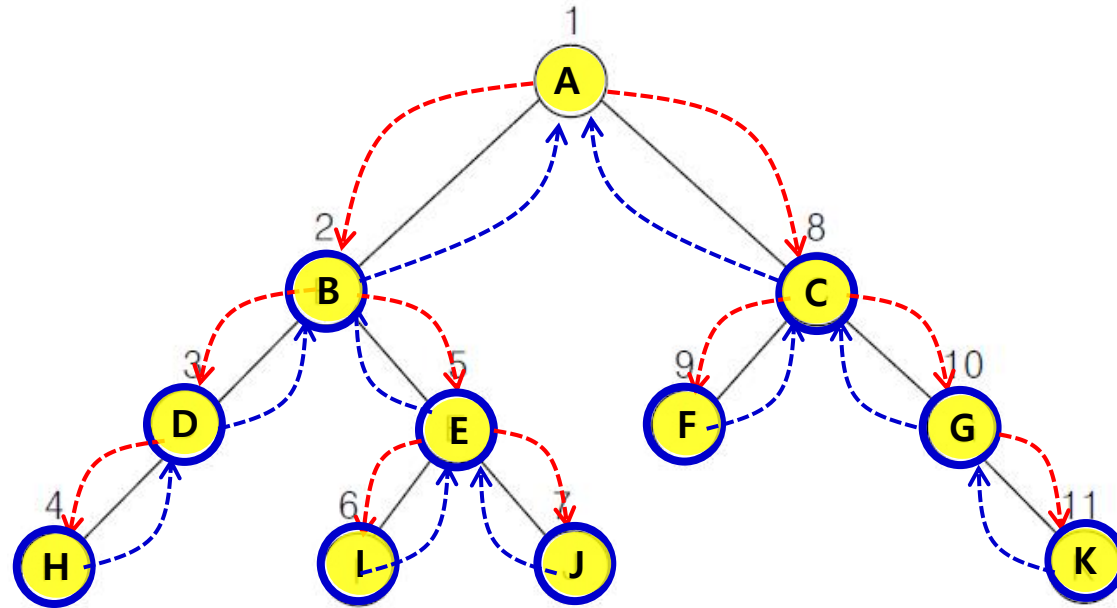
- 후위 순회 (L-R-D)의 예
- 이진트리의 후위 순회 경로: H-D-I-J-E-B-F-K-G-C-A



[그림 8-18] 이진 트리의 후위 순회 경로: H-D-I-J-E-B-F-K-G-C-A

# Tree

- 후위 순회 과정 -> H-D-I-J-E-B-F-K-G-C-A



① 노드 A(ⓁRD) → 노드 B - 루트 A에서 후위 순회를 시작하여 노드 A의 왼쪽 서브트리 B로 이동한다.

노드 A(ⓁRD) → 노드 B(ⓁRD) → 노드 D - 현재 노드 B에서 왼쪽 서브트리 D로 이동하여,

노드 A(ⓁRD) → 노드 B(ⓁRD) → 노드 D(ⓁRD) → 노드 H - 현재 노드 D의 왼쪽 단말 노드 H의 데이터를 읽는다.

# Tree

- 후위 순회 과정 -> H-D-I-J-E-B-F-K-G-C-A

② 노드 A(LRD) → 노드 B(LRD) → 노드 D(LRD) → 공백 노드 - 노드 D의 오른쪽 단말 노드인 공백 노드를 읽고

노드 A(LRD) → 노드 B(LRD) → 노드 D(LRD) - 현재 노드 D의 데이터를 읽는다.

노드 A(LRD) → 노드 B(LRD) ← 노드 D(LRD) - 노드 D에서의 LRD 작업이 끝났으므로 이전 경로인 노드 B로 돌아간다.

③ 노드 A(LRD) → 노드 B(LRD) → 노드 E - 현재 노드 B의 오른쪽 서브트리 E로 이동하여

노드 A(LRD) → 노드 B(LRD) → 노드 E(LRD) → 노드 I - 현재 노드 E의 왼쪽 단말 노드 I의 데이터를 읽는다.

④ 노드 A(LRD) → 노드 B(LRD) → 노드 E(LRD) → 노드 J - 노드 E의 오른쪽 단말 노드 J의 데이터를 읽는다.

⑤ 노드 A(LRD) → 노드 B(LRD) → 노드 E(LRD) - 현재 노드 E의 데이터를 읽는다.

노드 A(LRD) → 노드 B(LRD) ← 노드 E(LRD) - 현재 노드 E에서의 LRD 작업이 끝났으므로, 이전 경로인 노드 B로 돌아간다.

# Tree

- 후위 순회 과정 -> H-D-I-J-E-B-F-K-G-C-A

⑥ 노드 A(LRD) → 노드 B(LRⓇ) - 현재 노드 B의 데이터를 읽고,  
노드 A(LRD) ← 노드 B(LRⓇ) - 현재 노드 B에서의 LRD 작업이 끝났으므로, 이전 경로인 노드 A로 돌아간다.  
노드 A(LRⓇ) → 노드 C - 현재 노드 A의 오른쪽 서브트리 C로 이동한다.

⑦ 노드 A(LRⓇ) → 노드 C(LRD) → 노드 F - 현재 노드 C의 왼쪽 단말 노드 F의 데이터를 읽는다.

⑧ 노드 A(LRⓇ) → 노드 C(LRⓇ) → 노드 G - 현재 노드 C의 오른쪽 서브트리 G로 이동하여,  
노드 A(LRⓇ) → 노드 C(LRⓇ) → 노드 G(LRD) → 공백 노드 - 현재 노드 G의 왼쪽 단말 노드인 공백 노드를 읽고,  
노드 A(LRⓇ) → 노드 C(LRⓇ) → 노드 G(LRⓇ) → 노드 K - 노드 G의 오른쪽 단말 노드 K의 데이터를 읽는다.

⑨ 노드 A(LRⓇ) → 노드 C(LRⓇ) → 노드 G(LRⓇ) - 현재 노드 G의 데이터를 읽고,  
노드 A(LRⓇ) → 노드 C(LRⓇ) ← 노드 G(LRⓇ) - 현재 노드 G에서의 LRD 작업이 끝났으므로 이전 경로인 노드 C로 돌아간다.

# Tree

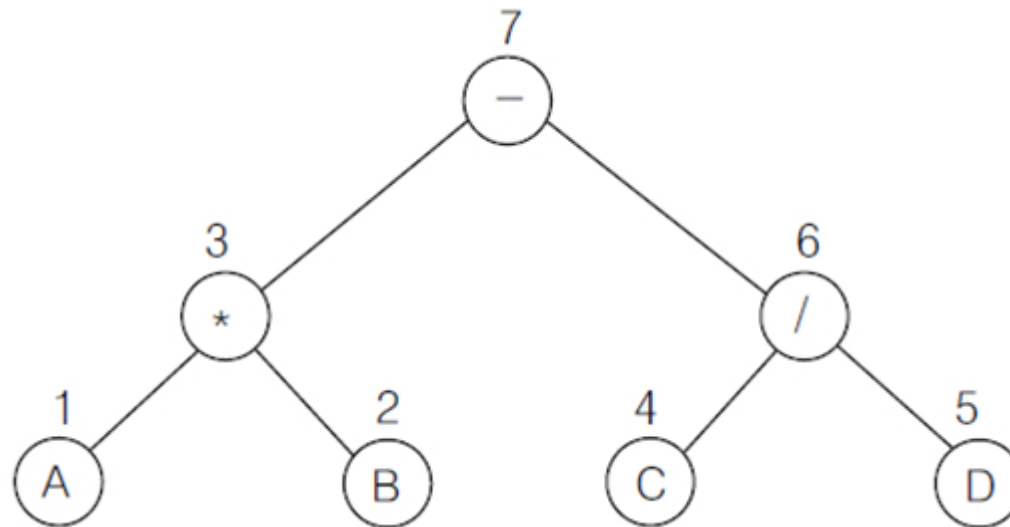
- 후위 순회 과정 -> H-D-I-J-E-B-F-K-G-C-A

- ⑩ **노드 A(LⓇD) → 노드 C(LⓇD)** - 현재 노드 C의 데이터를 읽고,  
**노드 A(LⓇD) ← 노드 C(LⓇD)** - 현재 노드 C에서의 LRD 작업이 끝났으므로 이전 경로인  
노드 A로 이동한다.
- ⑪ **노드 A(LⓇD)** - 현재 노드 A의 데이터를 읽는다. 이로써 루트 노드 A에 대한 LRD 순회가 끝  
났으므로 트리 전체에 대한 후위 순회가 완성되었다.

# Tree

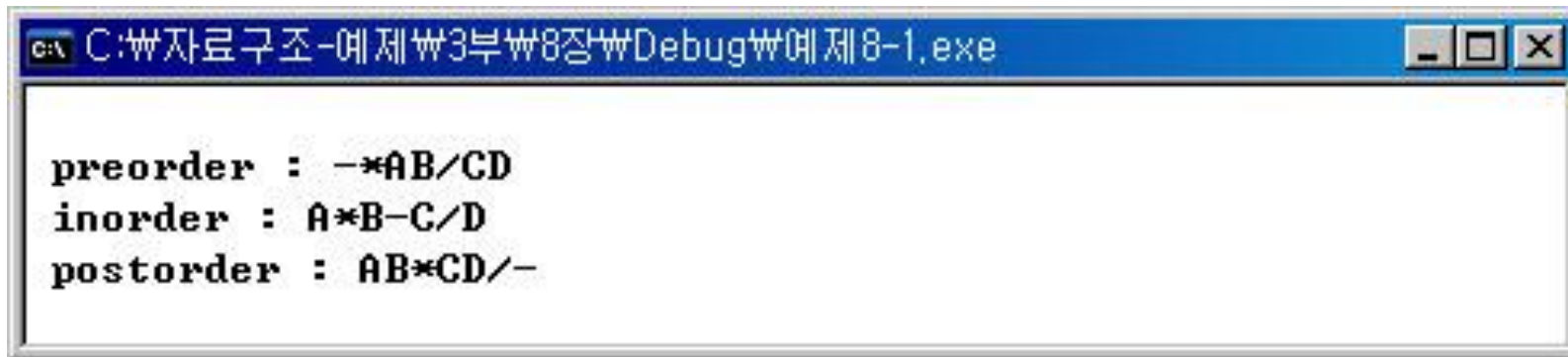
## – 수식 이진 트리에 대한 후위 순회

- 수식 이진 트리를 후위 순회하면, 수식에 대한 후위 표기식을 구할 수 있다.
- [그림 8-15]의 수식 이진 트리의 후위 순회 경로 >> **AB\*CD/-**



# Tree

- 이진 트리에서 순회 프로그램 작성하기 - 실습 1
  - Preorder : 전위 순회
  - Inorder : 중위 순회
  - Postorder : 후위 순회



```
C:\자료구조-예제\3부\8장\Debug\예제8-1.exe

preorder : -*AB/CD
inorder : A*B-C/D
postorder : AB*CD/-
```

# Tree

- 후위 순회의 예 ) 하위 폴더의 용량을 계산하여 상위 폴더의 용량과 더해서 전체 용량을 계산해야 하므로 후위 순회를 이용
  - **프로그램 폴더의 전체 용량** = 프로그램 폴더(2M) + 하위 폴더의 용량{C프로그램 폴더(200M) + Java프로그램 폴더(100M)} = 302M
  - **C:\#의 전체 용량** = C:\#의 용량(0M) + 하위 폴더의 용량{프로그램 폴더의 전체 용량(302M) + 자료 폴더(15M)} = 317M
  - **그림 폴더의 전체 용량** = 그림 폴더(68M) + 하위 폴더의 용량{사진 폴더(55M) + 동영상 폴더(120M)} = 243M
  - **D:\#의 전체 용량** = D:\#의 용량(10M) + 하위 폴더의 용량{음악 폴더의 용량(40M) + 그림 폴더의 전체 용량(243M)} = 293M
  - **내 컴퓨터의 전체 용량** = 내 컴퓨터의 용량(0M) + 하위 폴더의 용량{C:\# 폴더의 전체 용량(317M) + D:\# 폴더의 전체 용량(293M)} = 610M



# Tree

