



7 트리

IT CookBook, C로 배우는 쉬운 자료구조(개정 3판)

❖ 학습목표

- 트리의 개념을 이해한다.
- 이진 트리의 자료구조를 알아본다.
- 이진 트리의 순회를 이해한다.
- 이진 탐색 트리의 개념을 이해하고 연산 방법을 이해한다.
- 힙의 자료구조를 이해한다.

❖ 내용

- 트리의 이해
- 이진 트리
- 이진 트리의 구현
- 이진 트리의 순회
- 이진 탐색 트리
- 힙의 개념과 연산 및 구현



1. 트리의 이해

❖ 트리(tree)

- 원소들 간에 1:n 관계를 가지는 비선형 자료구조
- 원소들 간에 계층관계를 가지는 계층형 자료구조(Hierarchical Data Structure)
- 상위 원소에서 하위 원소로 내려가면서 확장되는 트리(나무)모양의 구조



1. 트리의 이해

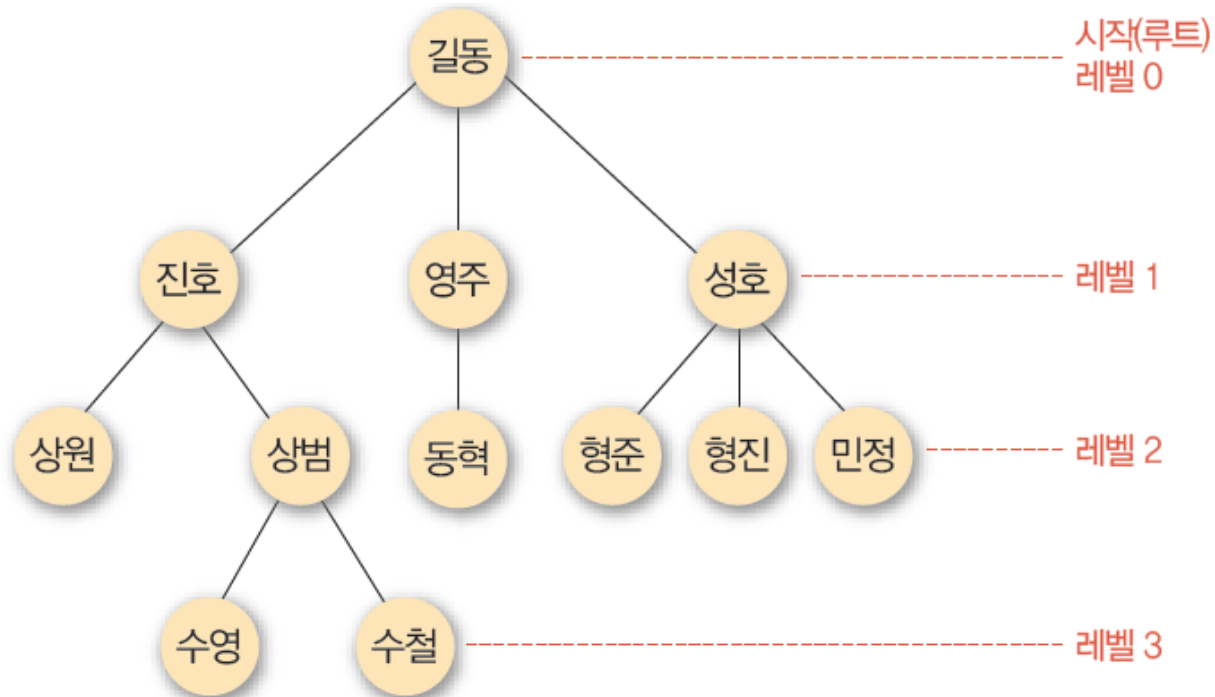


그림 7-1 트리 자료구조의 예 : 가계도

- 트리 자료구조의 예 - 가계도
 - 가계도의 자료 : 가족 구성원
 - 자료를 연결하는 선 : 부모-자식 관계 표현



1. 트리의 이해

- 길동의 자식 – 진호, 영주, 성호
- 진호, 영주, 성호의 부모 – 길동
- 같은 부모의 자식들끼리는 형제관계
 - 진호, 영주, 성호는 형제관계
- 조상 – 자신과 연결된 선을 따라 올라가면서 만나는 사람들
 - 수영의 조상 : 상범, 진호, 길동
- 자손 - 자신과 연결된 선을 따라 내려가면서 만나는 사람들
 - 진호의 자손 : 상원, 상범, 수영, 수철
- 가계도의 시작(루트)인 길동이를 0세대(레벨 0), 길동이 자식들은 1세대(레벨 1), 그 다음 자식들은 2세대(레벨 2), 그 다음 자식들은 3세대(레벨 3)
- 가족 구성원 누구든지 분가하여 독립된 가계를 이룰 수 있음



1. 트리의 이해

■ 트리 A

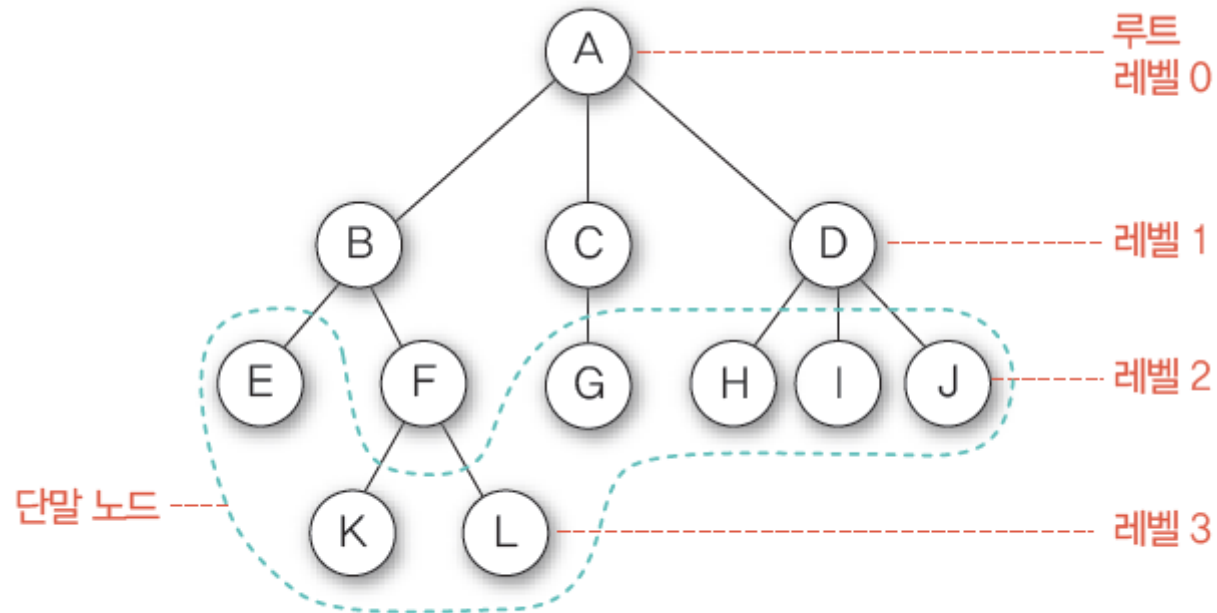


그림 7-2 트리의 구조와 구성 요소



1. 트리의 이해

■ 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(레벨^{Level} 0)
 - 트리 A의 루트노드 - A
- **간선(edge)** – 노드를 연결하는 선. 부모^{Parent} 노드와 자식^{Child} 노드를 연결
- **형제 노드(sibling node)** – 같은 부모 노드의 자식 노드들
 - B,C,D는 형제 노드
- **조상 노드(Ancessor)** – 간선을 따라 루트 노드까지 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 가짐
- **자손 노드** – 서브 트리에 있는 하위 레벨의 노드들
 - B의 자손 노드 - E,F,K,L



1. 트리의 이해

- 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**



1. 트리의 이해

- **포리스트 forest** : 서브트리의 집합

- 트리 A에서 노드 A를 제거하면, A의 자식 노드 B, C, D에 대한 서브 트리가 생기고, 이들의 집합은 포리스트가 됨

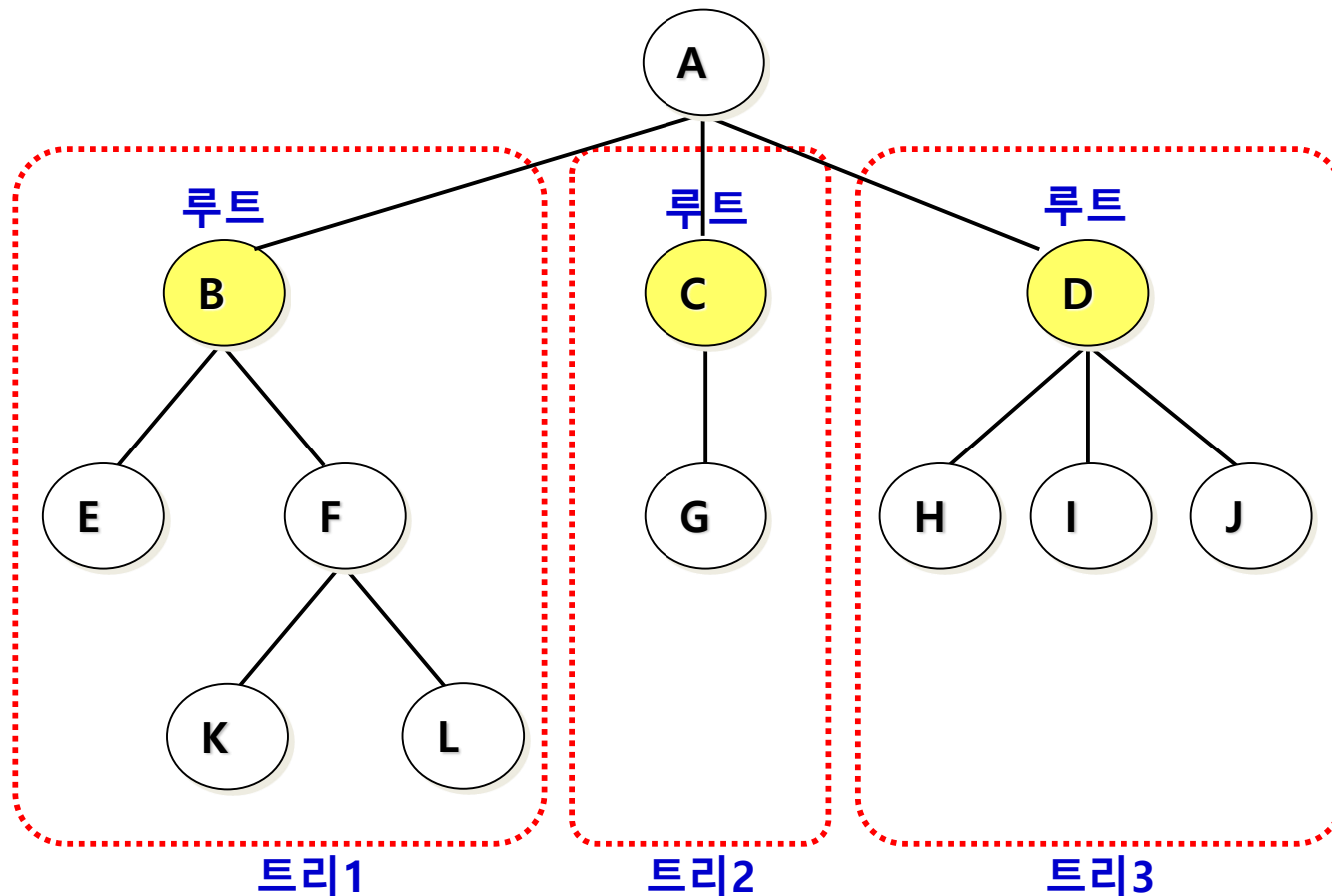


그림 7-3 [그림 7-2]에서 루트 노드 A를 제거하여 만든 포리스트



2. 이진트리 : 개념과 구조

❖ 이진트리(Binary Tree)

- 트리의 모든 노드의 차수를 2 이하로 제한하여 전체 트리의 차수가 2 이하가 되도록 정의
- 이진 트리의 모든 노드는 왼쪽 자식 노드와 오른쪽 자식 노드만 가짐
 - 부모 노드와 자식 노드 수와의 관계 ➡ 1:2
 - 공백 노드도 자식 노드로 취급
 - $0 \leq \text{노드의 차수} \leq 2$

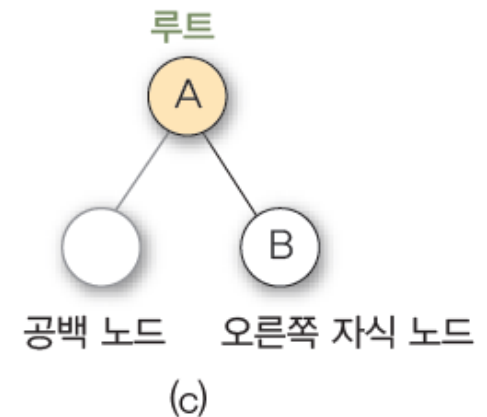
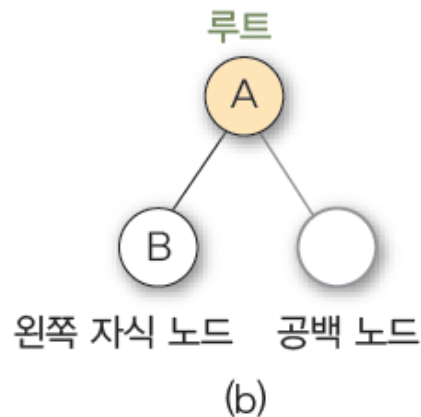
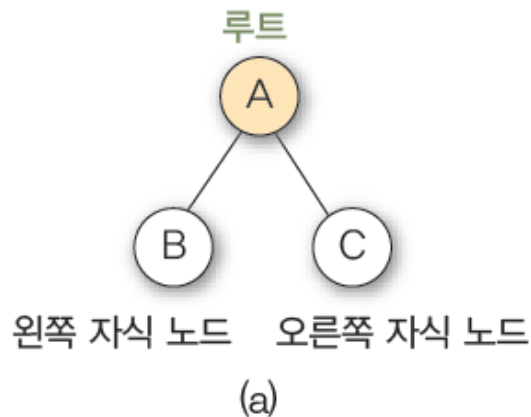


그림 7-4 이진 트리의 기본 구조

2. 이진트리 : 개념과 구조

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

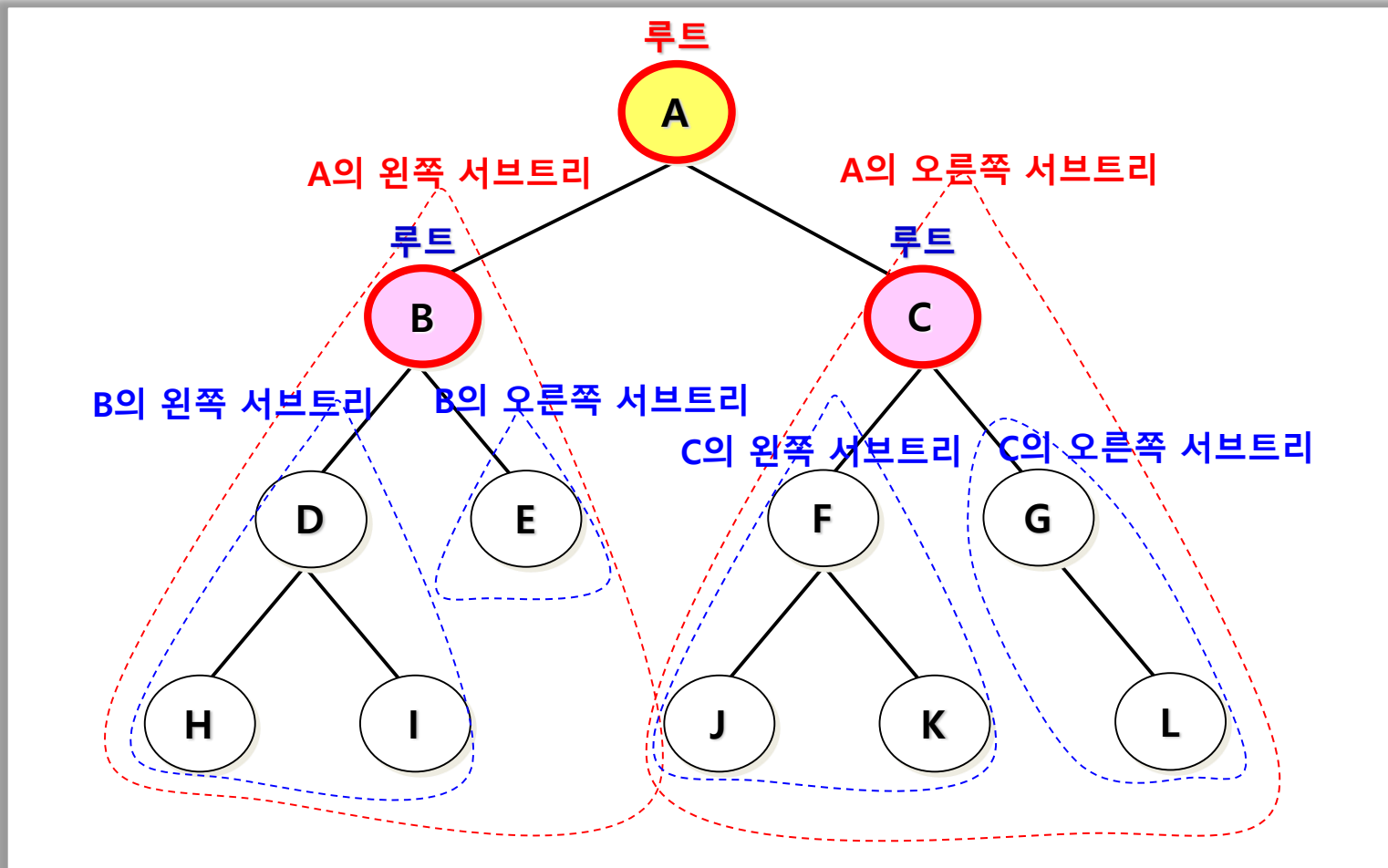
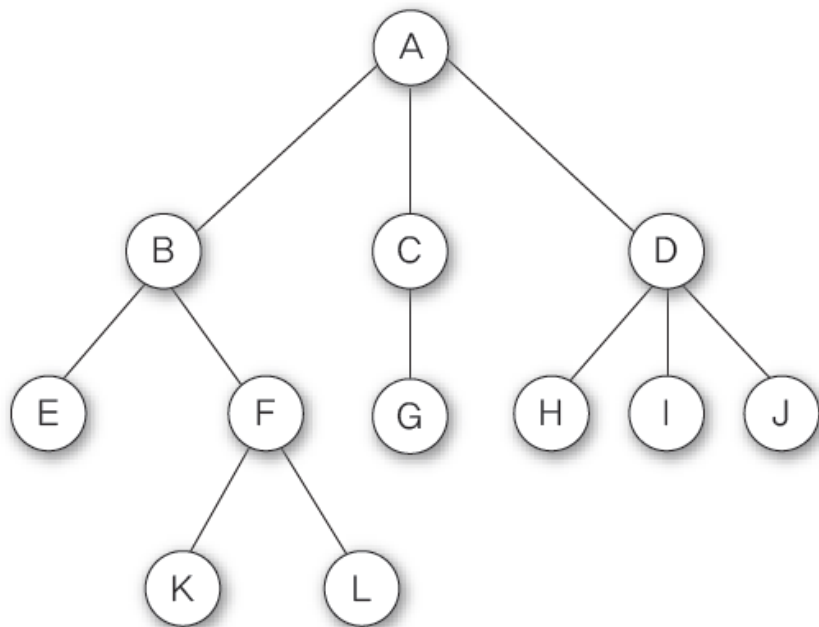


그림 7-5 이진 트리의 서브 트리

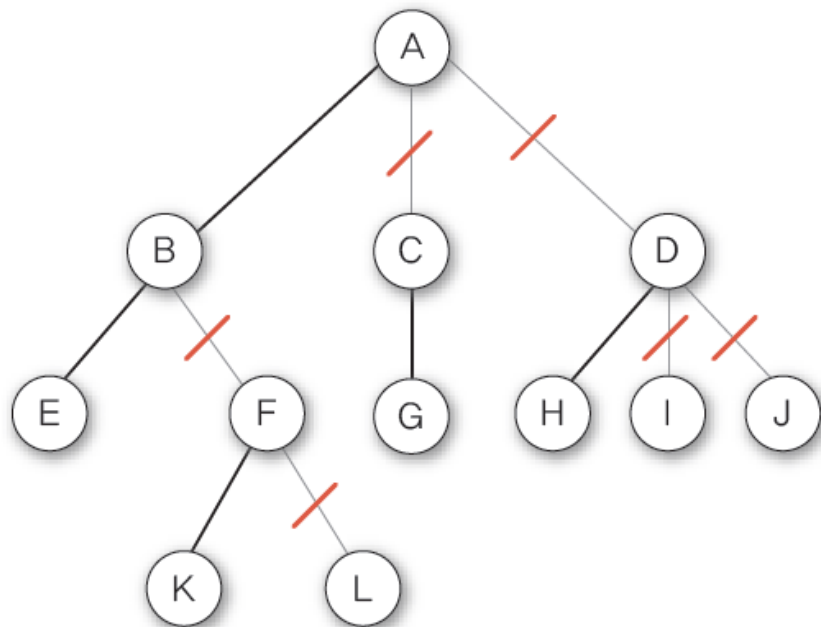
2. 이진트리 : 개념과 구조

■ 일반 트리를 이진 트리로 변환

① 일반 트리 A

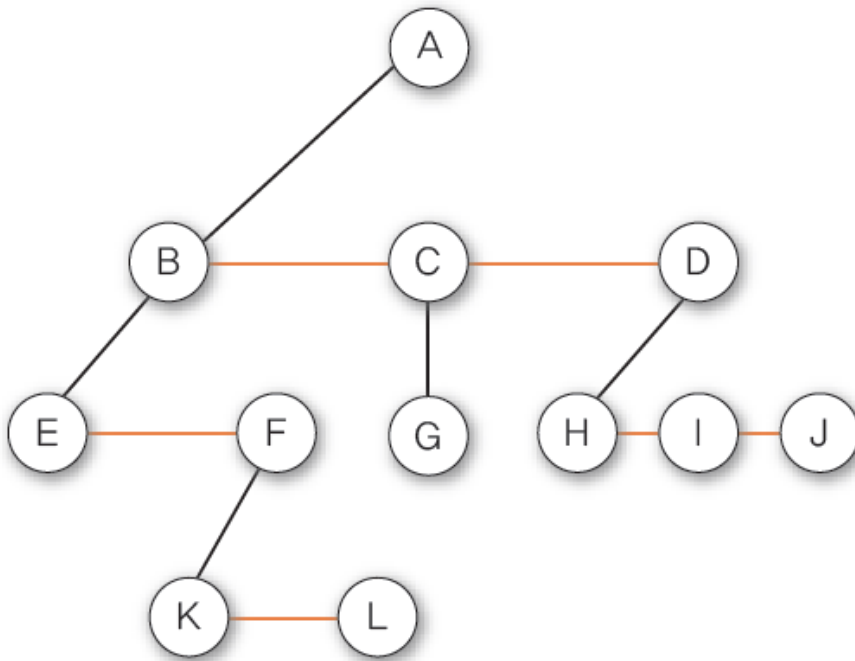


① 첫 번째 자식 노드 간선만 남기고 나머지 간선 제거



2. 이진트리 : 개념과 구조

② 형제 노드를 간선으로 연결



③ 시계 방향으로 45° 회전

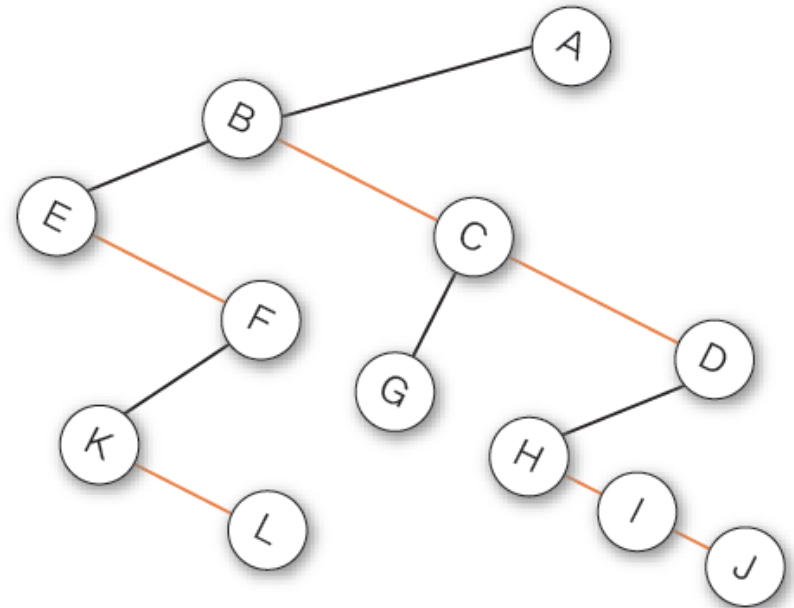


그림 7-6 일반 트리를 이진 트리로 변환하는 과정



2. 이진트리 : 추상 자료형

ADT 7-1

이진 트리의 추상 자료형

ADT BinaryTree

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

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

// 공백 이진 트리를 생성하는 연산

$\text{createBT}() ::= \text{create an empty binary tree};$

// 이진 트리가 공백인지 확인하는 연산

$\text{isEmpty}(bt) ::= \text{if } (bt \text{ is empty}) \text{ then return true}$
 $\text{else return false};$

// 두 개의 이진 서브 트리를 연결하여 하나로 만드는 연산

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



2. 이진트리 : 추상 자료형

// 이진 트리의 왼쪽 서브 트리를 구하는 연산

```
leftSubtree(bt) ::= if (isEmpty(bt)) then return NULL  
                  else return left subtree of bt;
```

// 이진 트리의 오른쪽 서브 트리를 구하는 연산

```
rightSubtree(bt) ::= if (isEmpty(bt)) then return NULL  
                   else return right subtree of bt;
```

// 이진 트리에서 루트 노드의 데이터(item)를 구하는 연산

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

End BinaryTree



2. 이진트리 : 추상 자료형

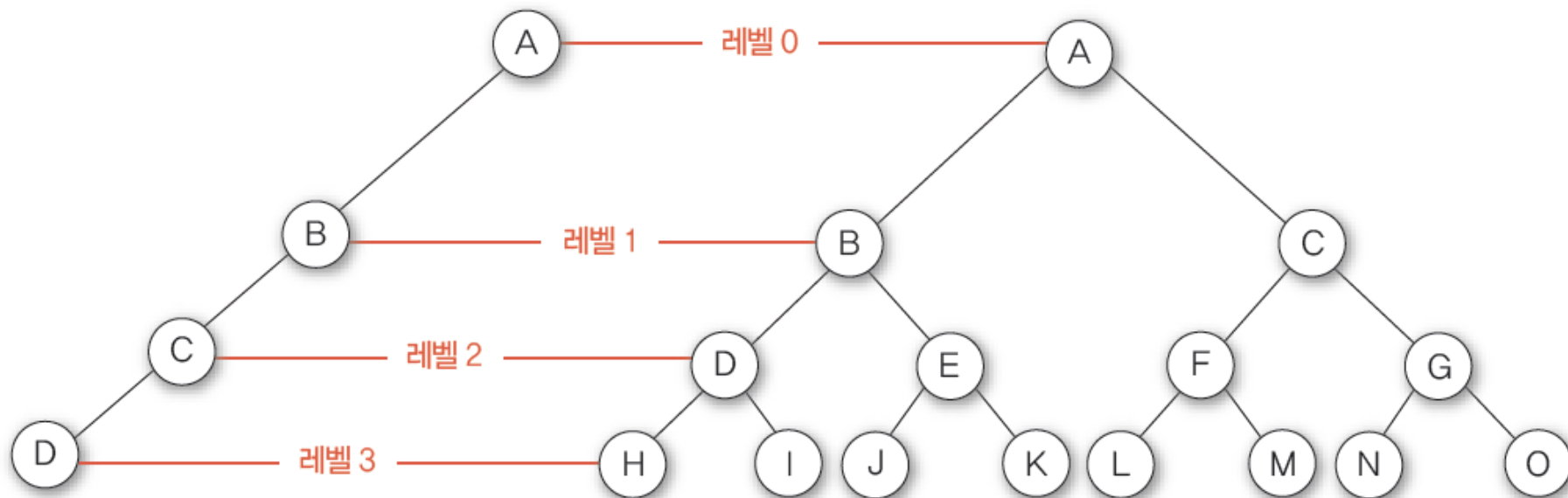
■ 이진트리의 특성

- 정의1) 노드가 n 개인 이진 트리는 항상 간선이 $(n-1)$ 개임
 - 루트를 제외한 $(n-1)$ 개의 노드가 부모 노드와 연결되는 한 개의 간선을 가짐
- 정의2) 높이가 h 인 이진 트리가 가질 수 있는 노드 개수는 최소 $(h+1)$ 개이며, 최대 $(2^{h+1}-1)$ 개
 - 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

$$\sum_{i=0}^h 2^i = 2^{h+1} - 1 \text{ 개}$$



2. 이진트리 : 추상 자료형



(a) 최소 노드를 갖는 이진 트리

(b) 최대 노드를 갖는 이진 트리

그림 7-7 높이가 3이면서 최소 노드를 갖는 이진 트리와 최대 노드를 갖는 이진 트리



2. 이진트리 : 종류

❖ 이진 트리의 종류

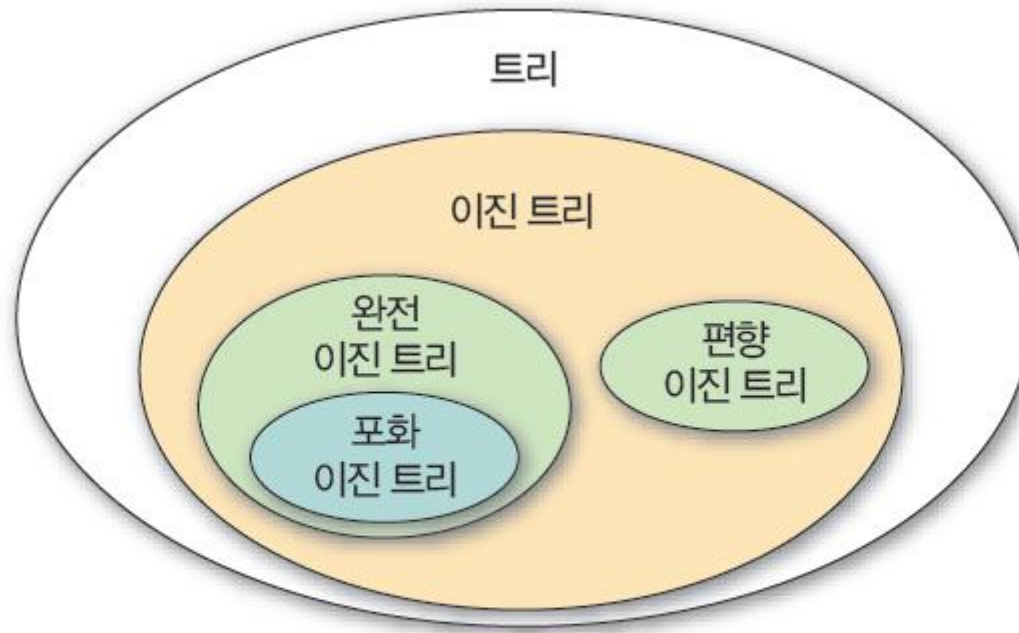


그림 7-8 이진 트리의 종류



2. 이진트리 : 종류

❖ 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

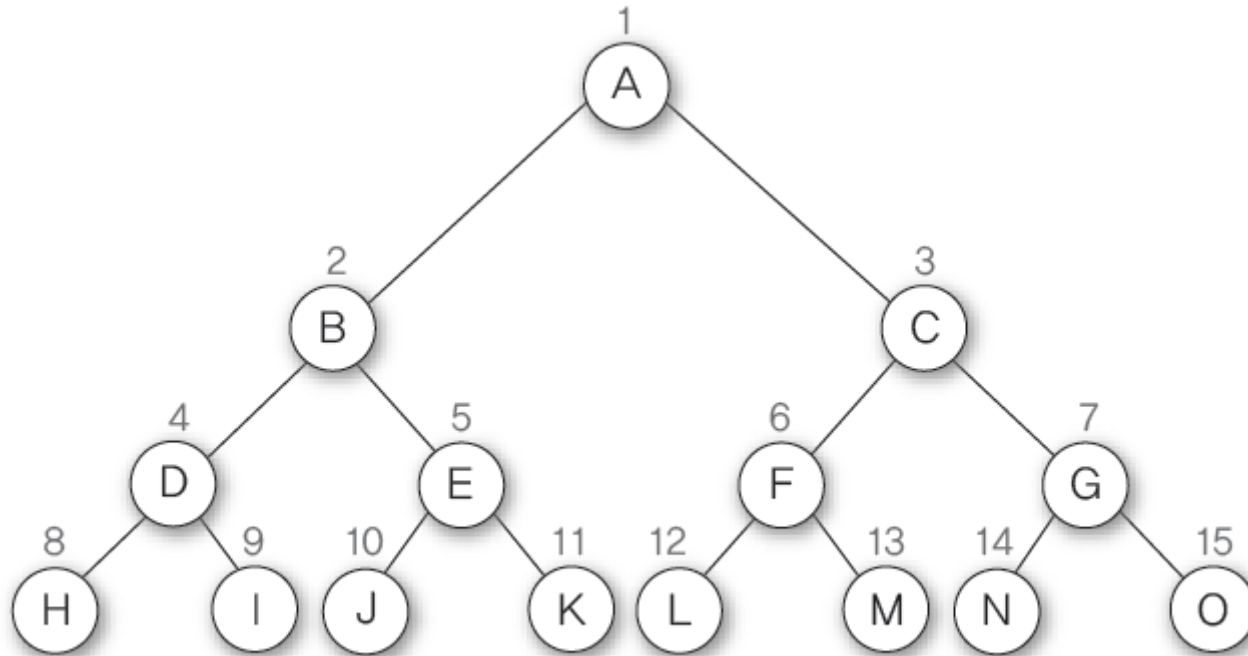


그림 7-9 높이가 3인 포화 이진 트리



2. 이진트리 : 종류

■ 완전 이진 트리 Complete Binary Tree

- 높이가 h 이고 노드 수가 n 개일 때 (단, $n < 2^{h+1}-1$), 노드 위치가 포화 이진 트리에서의 노드 1번부터 n 번까지의 위치와 완전히 일치하는 이진 트리
- 완전 이진 트리에서는 $(n+1)$ 번부터 $(2^{h+1}-1)$ 번까지 노드는 모두 공백 노드

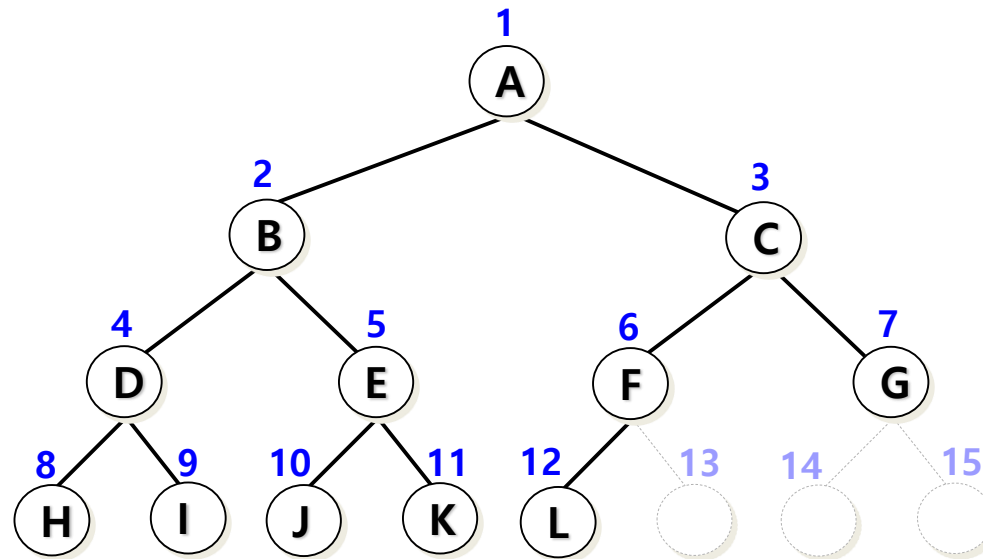
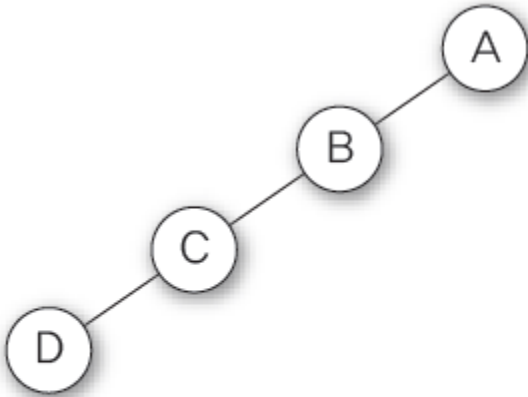


그림 7-10 높이가 3인 완전 이진 트리

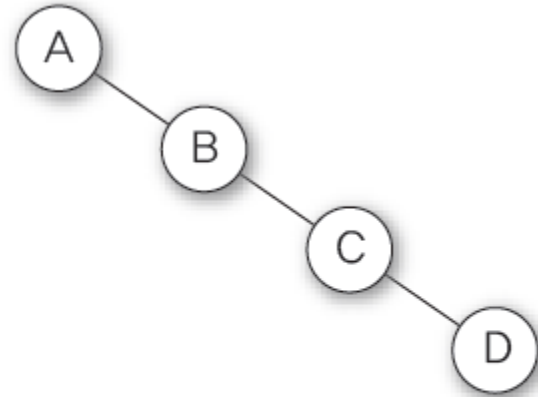


2. 이진트리 : 종류

- 편향 이진 트리 (Skewed Binary Tree)
 - 높이가 h 일 때 $h+1$ 개의 노드를 가지면서 모든 노드가 왼쪽이나 오른쪽 중 한 방향으로만 서브 트리를 가지고 있는 트리



(a) 왼쪽 편향 이진 트리



(b) 오른쪽 편향 이진 트리

그림 7-11 높이가 3인 편향 이진 트리



3. 이진트리 구현 : 순차 자료구조를 이용한 구현

❖ 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



3. 이진트리 구현 : 순차 자료구조를 이용한 구현

■ 완전 이진 트리의 1차원 배열 표현

표 7-1 이진 트리를 표현한 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	$n > 0$

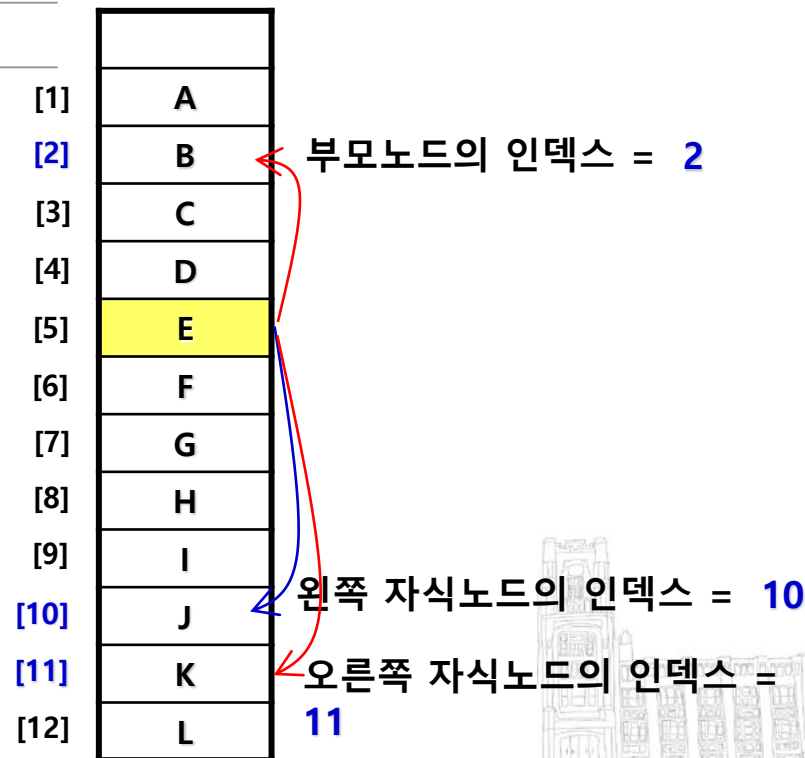
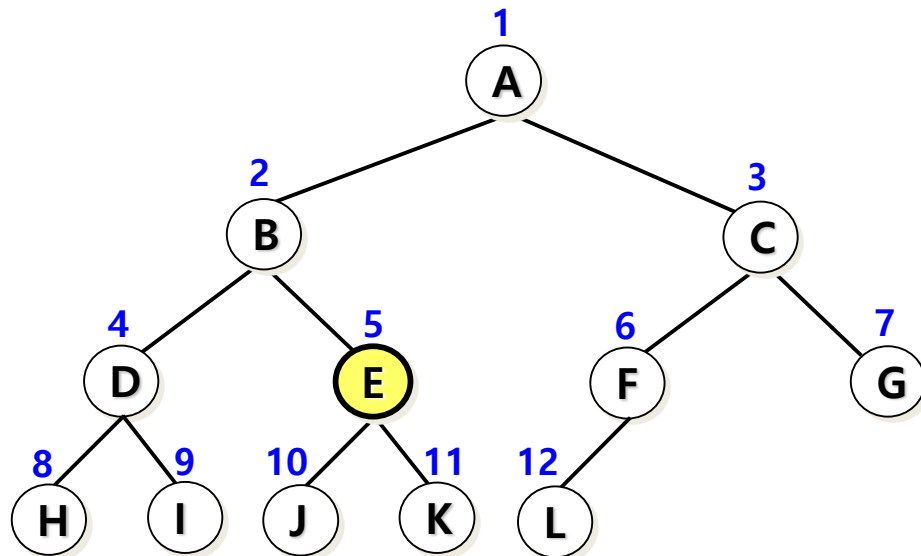
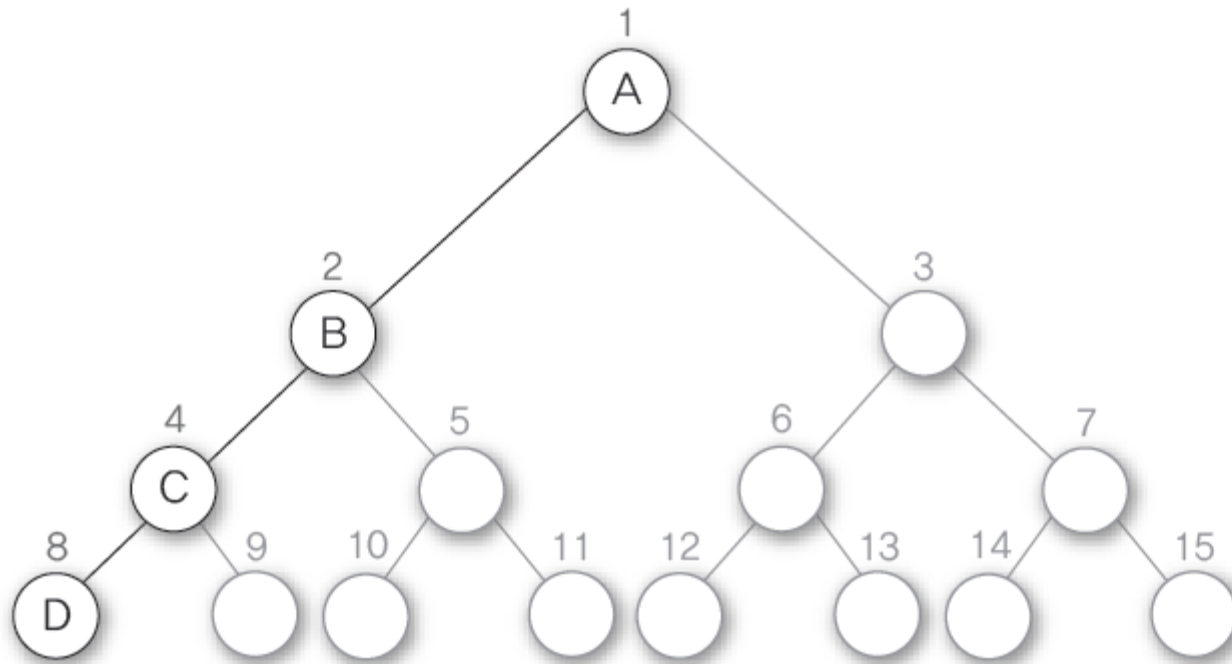


그림 7-12 이진 트리의 배열 표현 예 1 : 완전 이진 트리의 1차원 배열 표현

3. 이진트리 구현 : 순차 자료구조를 이용한 구현

- 편향 이진 트리의 1차원 배열 표현



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

그림 7-13 이진 트리의 배열 표현 예 2 : 편향 이진 트리의 1차원 배열 표현



3. 이진트리 구현 : 연결 자료구조를 이용한 구현

❖ 연결 자료구조를 이용한 이진트리의 구현

- 포인터를 사용하여 이진트리 구현
 - 데이터를 저장하는 데이터 필드, 왼쪽 자식 노드를 연결하는 왼쪽 링크 필드, 오른쪽 자식 노드를 연결하는 오른쪽 링크 필드로 구성. 자식 노드가 없으면 링크 필드에 NULL을 저장하여 NULL 포인터로 설정

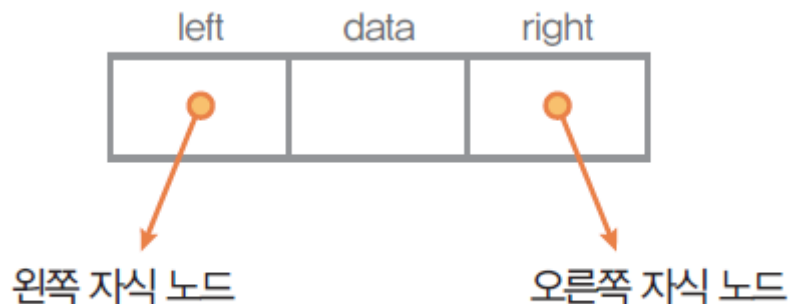


그림 7-14 이진 트리 노드의 구조

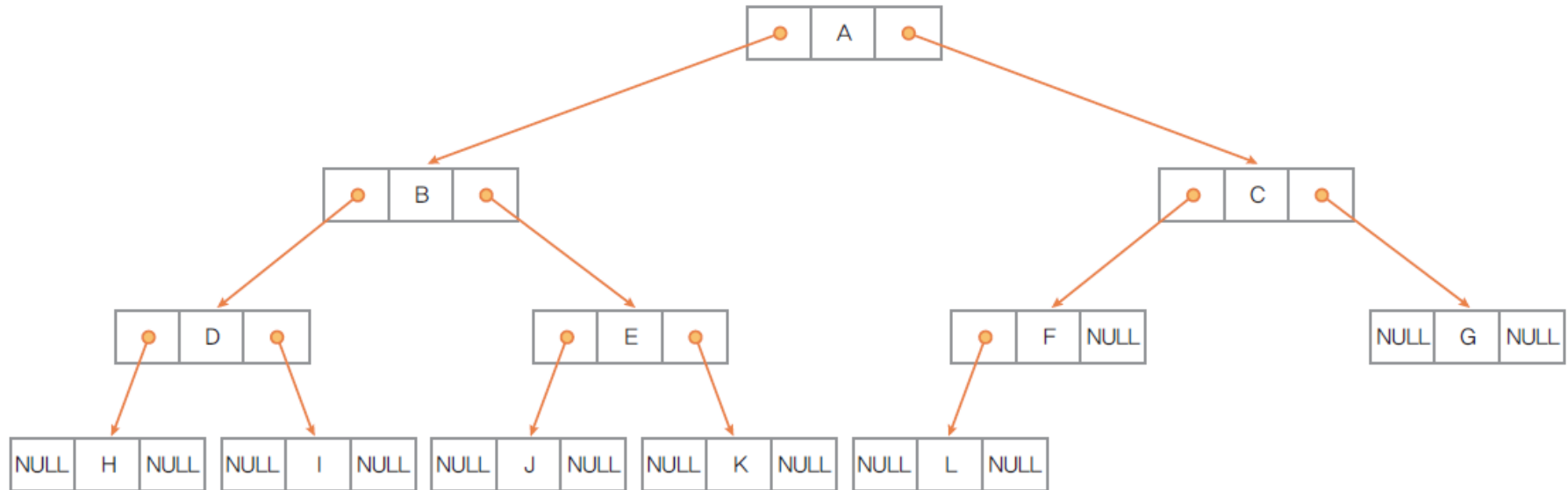
```
typedef struct treeNode {  
    char data;  
    struct treeNode *left;  
    struct treeNode *right;  
} treeNode;
```

그림 7-15 이진 트리 노드의 C 구조체 정의



3. 이진트리 구현 : 연결 자료구조를 이용한 구현

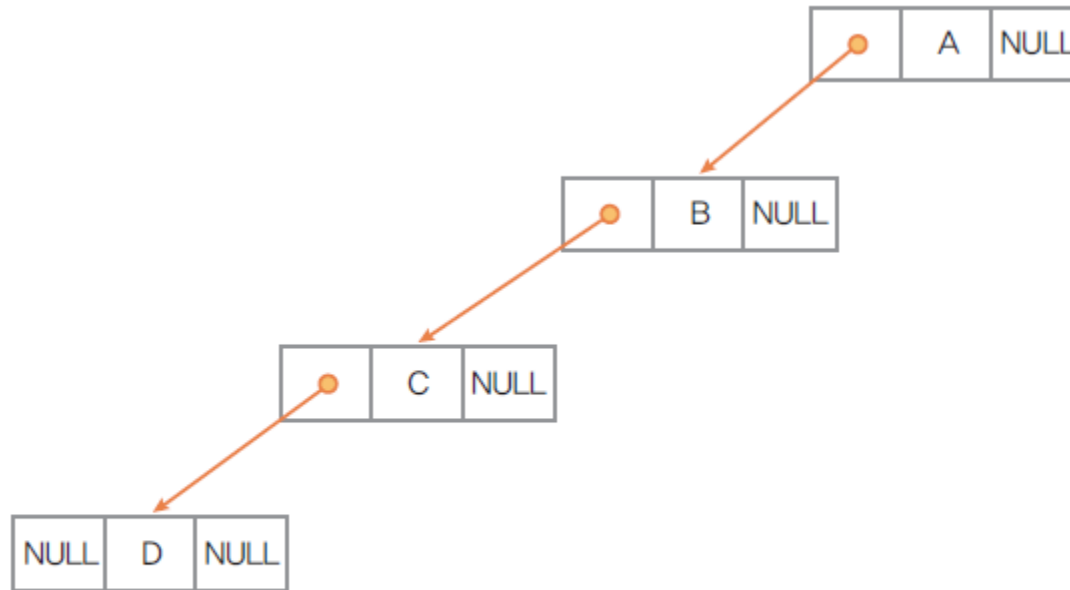
- 완전 이진 트리와 편향 이진 트리를 연결 자료구조 형태로 표현



(a) [그림 7-12] 완전 이진 트리에 대한 연결 자료구조 표현



3. 이진트리 구현 : 연결 자료구조를 이용한 구현



(b) [그림 7-13] 편향 이진 트리에 대한 연결 자료구조 표현

그림 7-16 이진 트리의 연결 자료구조 표현



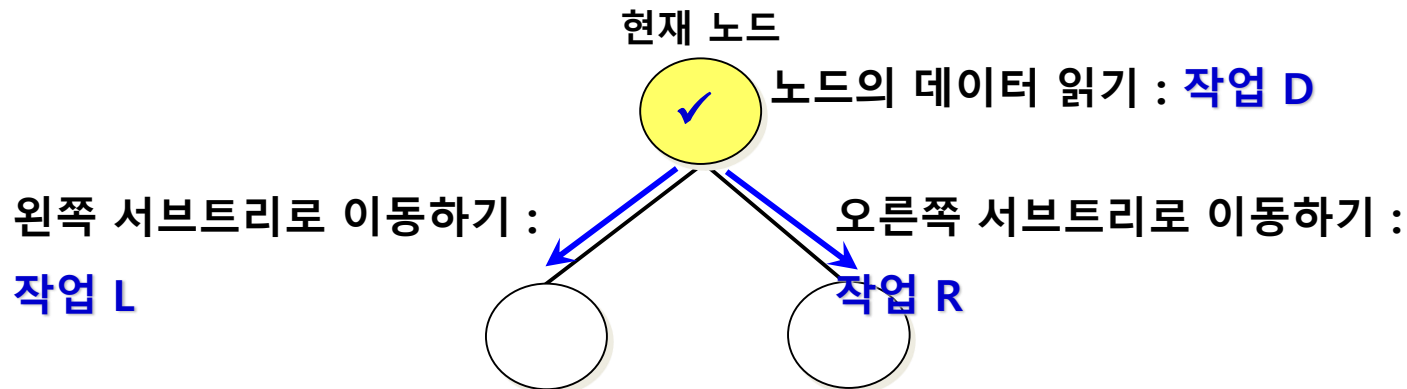
4. 이진 트리의 순회

❖ 이진 트리 순회의 개념

- 모든 원소를 빠트리거나 중복하지 않고 처리하는 연산

- 작업 D : 현재 노드를 방문하여 처리한다.
- 작업 L : 현재 노드의 왼쪽 서브 트리로 이동한다.
- 작업 R : 현재 노드의 오른쪽 서브 트리로 이동한다.

그림 7-17 이진 트리의 순회를 위한 세부 작업



4. 이진트리의 순회

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



4. 이진 트리의 순회

❖ 전위 순회 preorder traversal

- $D \rightarrow L \rightarrow R$ 순서로, 현재 노드를 방문하여 처리하는 작업 D를 가장 먼저 수행

- ① 작업 D : 현재 노드 n 을 처리한다.
- ② 작업 L : 현재 노드 n 의 왼쪽 서브 트리로 이동한다.
- ③ 작업 R : 현재 노드 n 의 오른쪽 서브 트리로 이동한다.

그림 7-18 이진 트리의 전위 순회 작업 순서

알고리즘 7-1 이진 트리의 전위 순회

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

4. 이진 트리의 순회

■ 전위 순회의 예

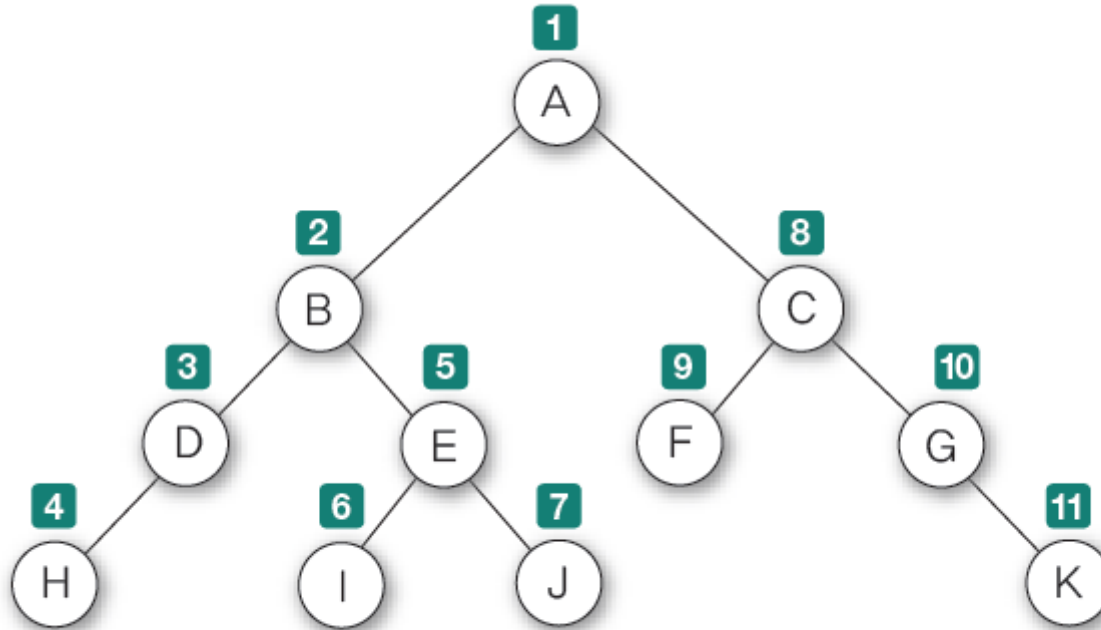


그림 7-19 이진 트리의 전위 순회 경로 : A-B-D-H-E-I-J-C-F-G-K



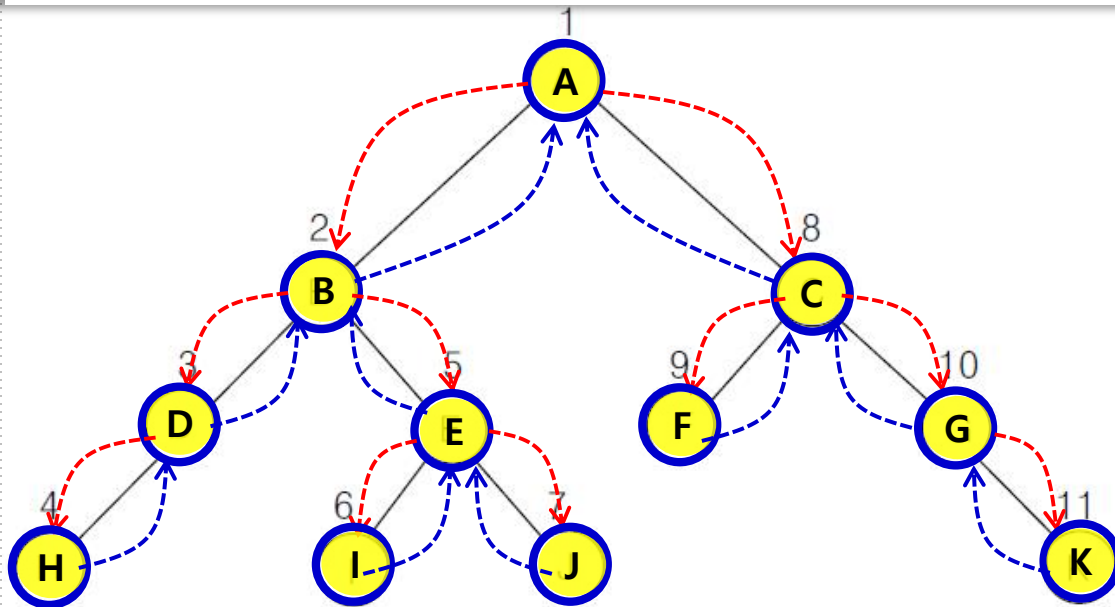
4. 이진트리의 순회

- 전위 순회 과정 >> A-B-D-H-E-I-J-C-F-G-K

노드 A(①②③) → 노드 B(①②③) ← 노드 ~~G(①②③)~~ - 이로써 노드 G에서의 DLR 순회가 끝났으므로 이전 노드 C로 돌아간다.

노드 A(①②③) ← 노드 ~~C(①②③)~~ - 현재 노드 C에서의 DLR 순회 역시 끝났으므로 다시 이전 노드 A로 돌아간다.

노드 A(①②③) - 이로써 루트 노드 A에 대한 DLR 순회가 끝났으므로 트리 전체에 대한 전위 순회가 완성되었다.



4. 이진 트리의 순회

- 수식 $A*B-C/D$ 를 이진 트리로 구성
 - 수식에 대한 이진 트리를 전위 순회하면, 전위 표기식을 구할 수 있음

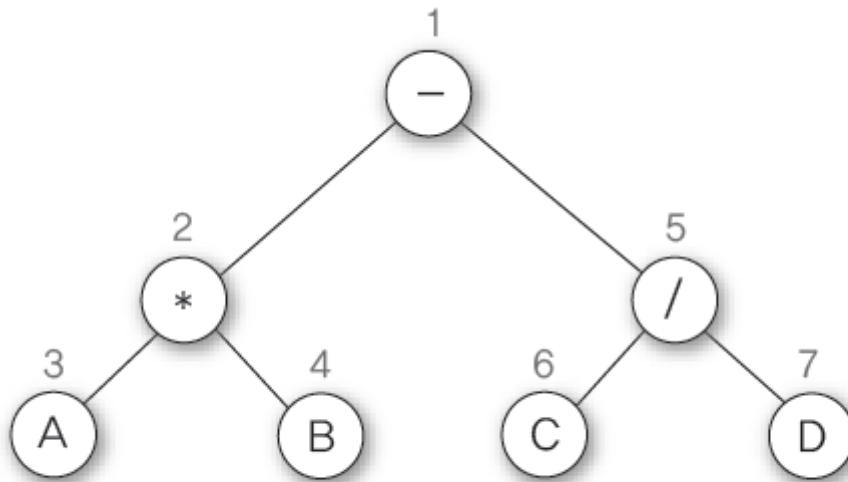


그림 7-20 수식에 대한 이진 트리의 전위 순회 경로: $-*AB/CD$



4. 이진 트리의 순회

❖ 중위 순회 inorder traversal

- $L \rightarrow D \rightarrow R$ 순서로, 현재 노드를 방문하는 작업 D를 작업 L과 작업 R의 중간에 수행

- ① 작업 L : 현재 노드 n의 왼쪽 서브 트리로 이동한다.
- ② 작업 D : 현재 노드 n을 처리한다.
- ③ 작업 R : 현재 노드 n의 오른쪽 서브 트리로 이동한다.

그림 7-21 이진 트리의 중위 순회 작업 순서

알고리즘 7-2 이진 트리의 중위 순회

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

4. 이진 트리의 순회

■ 중위 순회의 예

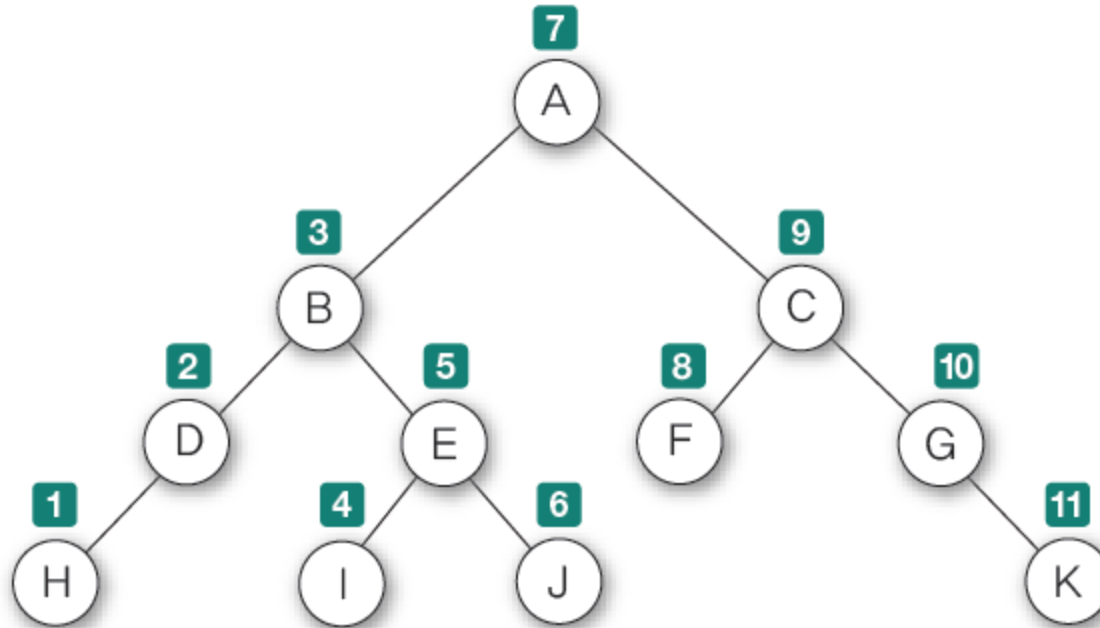


그림 7-22 이진 트리의 중위 순회 경로 : H-D-B-I-E-J-A-F-C-G-K



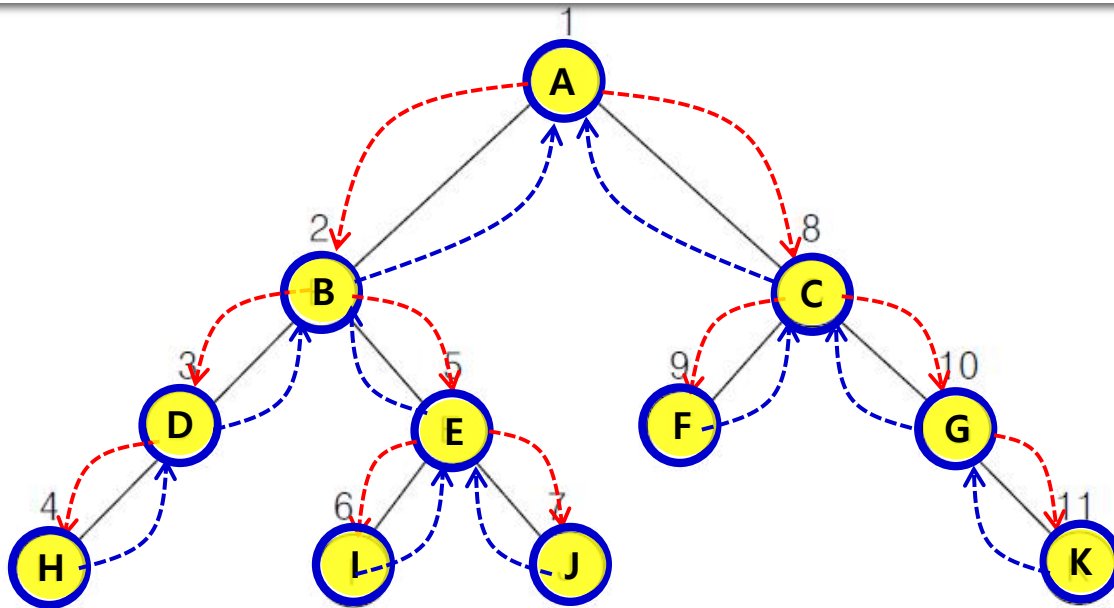
4. 이진트리의 순회

- 중위 순회 과정 >> H-D-B-I-E-J-A-F-C-G-K

노드 A($\textcircled{L}\textcircled{D}\textcircled{R}$) \rightarrow 노드 C($\textcircled{L}\textcircled{D}\textcircled{R}$) \leftarrow ~~노드 G($\textcircled{L}\textcircled{D}\textcircled{R}$)~~ - 노드 G에서의 LDR 순회가 끝났으므로 이전 노드 C로 돌아간다.

노드 A($\textcircled{L}\textcircled{D}\textcircled{R}$) \leftarrow ~~노드 C($\textcircled{L}\textcircled{D}\textcircled{R}$)~~ - 현재 노드 C에서의 LDR 순회 역시 끝났으므로 다시 이전 노드 A로 돌아간다.

~~노드 A($\textcircled{L}\textcircled{D}\textcircled{R}$)~~ - 이로써 루트 노드 A에 대한 LDR 순회가 모두 끝났으므로 트리 전체에 대한 중위 순회가 완성되었다.



4. 이진 트리의 순회

- 수식 $A*B-C/D$ 를 이진 트리로 구성
 - 수식 이진 트리를 중위 순회하면, 수식에 대한 중위 표기식을 구할 수 있음

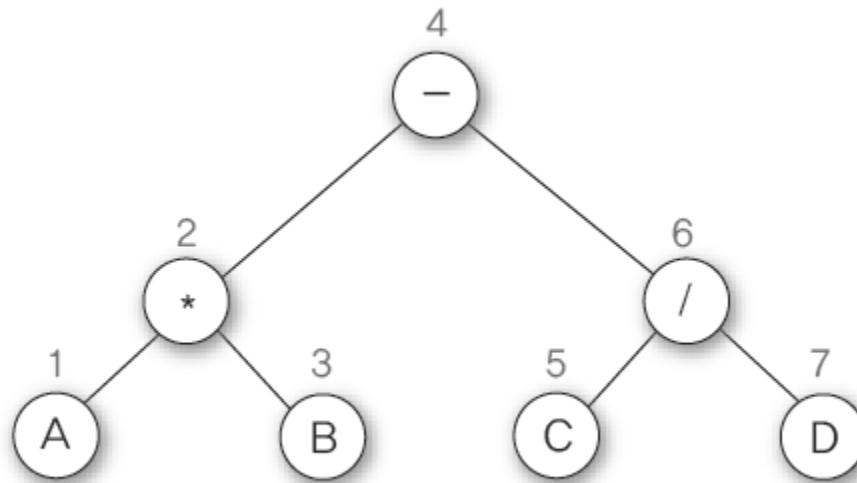


그림 7-23 수식에 대한 이진 트리의 중위 순회 경로 : $A*B-C/D$



4. 이진 트리의 순회

❖ 후위 순회 postorder traversal

- L-R-D 순서로 현재 노드를 방문하는 D 작업을 가장 나중에 수행

- ❶ 작업 L : 현재 노드 n 의 왼쪽 서브 트리로 이동한다.
- ❷ 작업 R : 현재 노드 n 의 오른쪽 서브 트리로 이동한다.
- ❸ 작업 D : 현재 노드 n 을 처리한다.

그림 7-24 이진 트리의 후위 순회 작업 순서

알고리즘 7-3 이진 트리의 후위 순회

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

4. 이진 트리의 순회

■ 후위 순회의 예

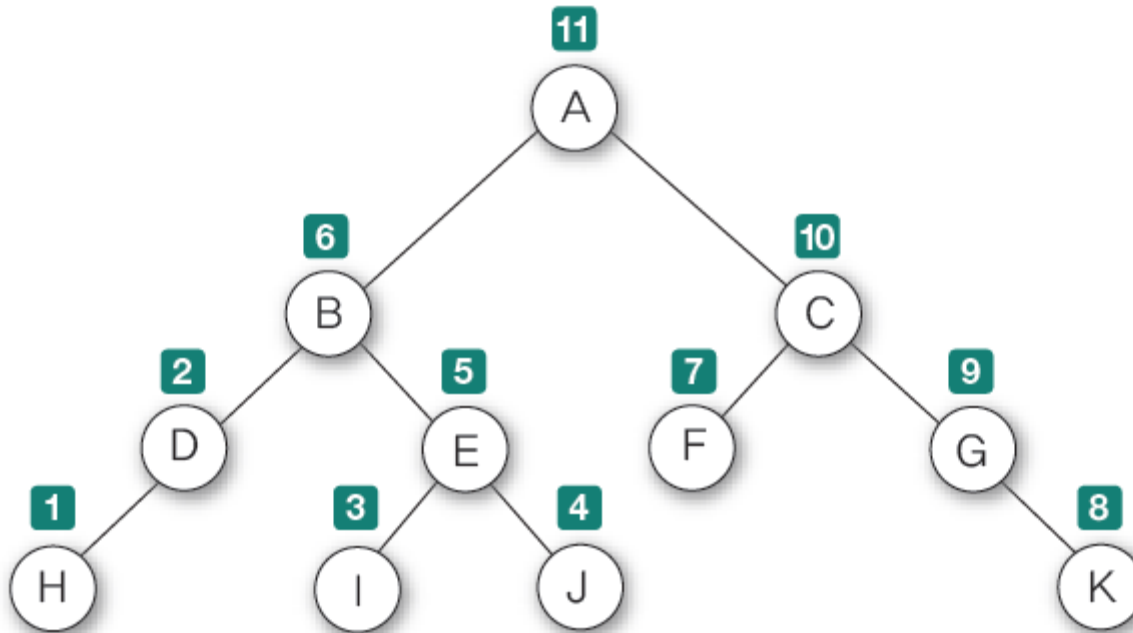


그림 7-25 이진 트리의 후위 순회 경로 : H-D-I-J-E-B-F-K-G-C-A

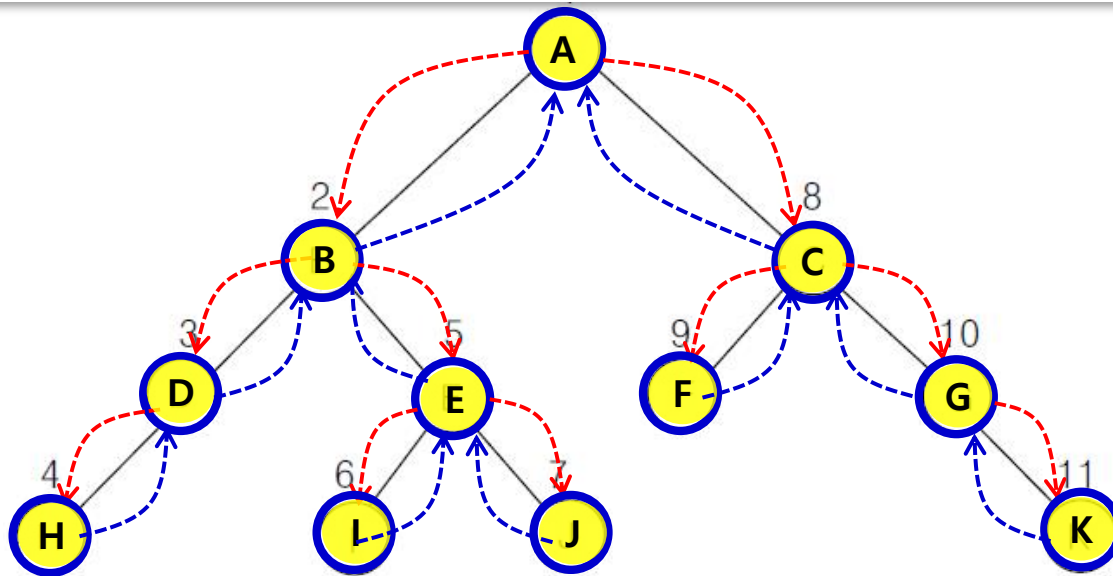


4. 이진트리의 순회

■ 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 끝났으므로 트리 전체에 대한 후위 순회가 완성되었다.

의 데이터를 읽는다.



4. 이진 트리의 순회

- 수식 $A*B-C/D$ 를 이진 트리로 구성
 - 수식 이진 트리를 후위 순회하면, 수식에 대한 후위 표기식을 구할 수 있음

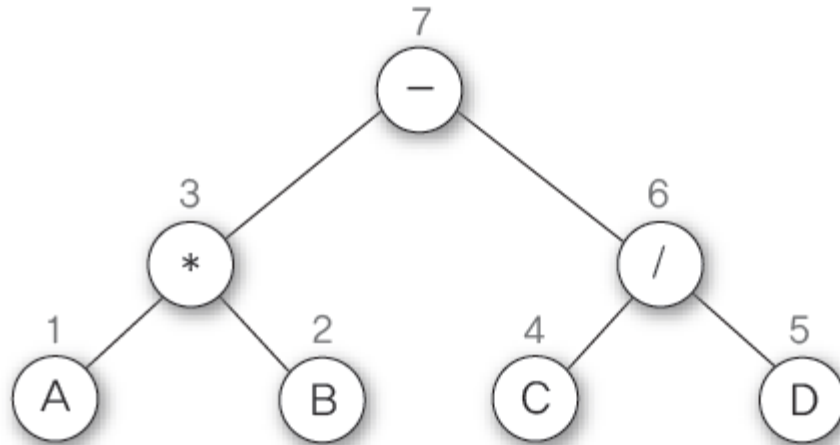
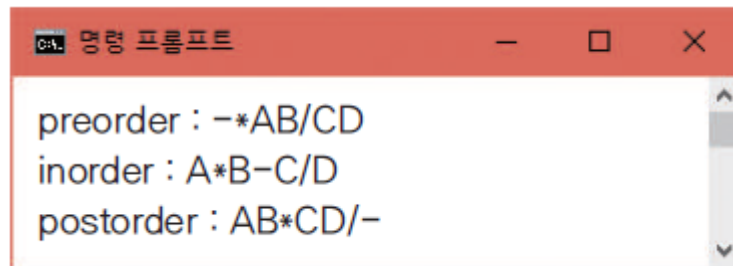


그림 7-26 수식에 대한 이진 트리의 후위 순회 경로 : $AB*CD/-$



4. 이진 트리의 순회

- 이진 트리 순회하기 프로그램 : [교재 335p](#)
 - 수식 ($A*B-C/D$)에 대한 이진 트리를 연결 자료구조 방식으로 표현
 - makeRootNode()함수를 이용해 전위 순회, 중위 순회, 후위 순회를 수행한 경로를 출력하여 확인하는 프로그램
- 실행 결과

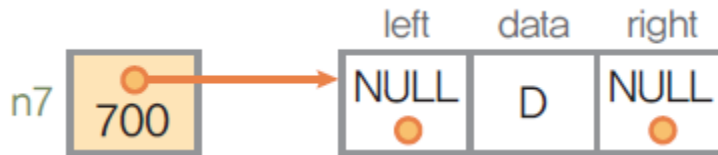


```
명령 프롬프트
preorder : -*AB/CD
inorder : A*B-C/D
postorder : AB*CD/-
```

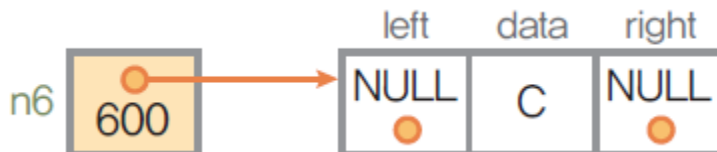


4. 이진 트리의 순회

- 11~18행 : data를 루트 노드로 하여 왼쪽 서브 트리와 오른쪽 서브 트리를 연결하는 연산을 수행
- 20~27행 : 매개변수 root를 시작으로 하는 전위 순회 연산을 수행
- 29~36행 : 매개변수 root를 시작으로 하는 중위 순회 연산을 수행
- 38~45행 : 매개변수 root를 시작으로 하는 후위 순회 연산을 수행
- 48~55행 : $A*B-C/D$ 의 수식에 대한 이진 트리를 구성
 - 49행 : 데이터 필드가 'D'이고 왼쪽 서브 트리과 오른쪽 서브 트리가 공백 노드인 이진 트리 노드 n7을 선언

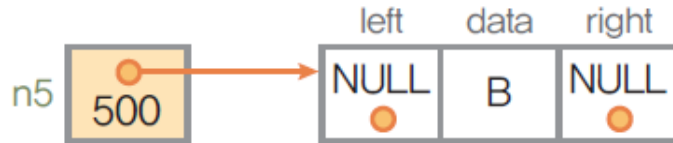


- 50행 : 데이터 필드가 'C'이고 왼쪽 서브 트리과 오른쪽 서브 트리가 공백 노드인 이진 트리 노드 n6 을 만들

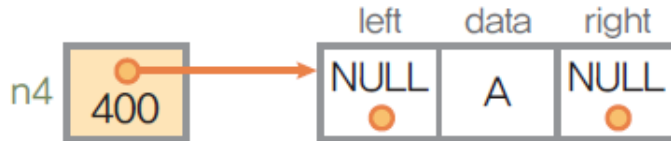


4. 이진 트리의 순회

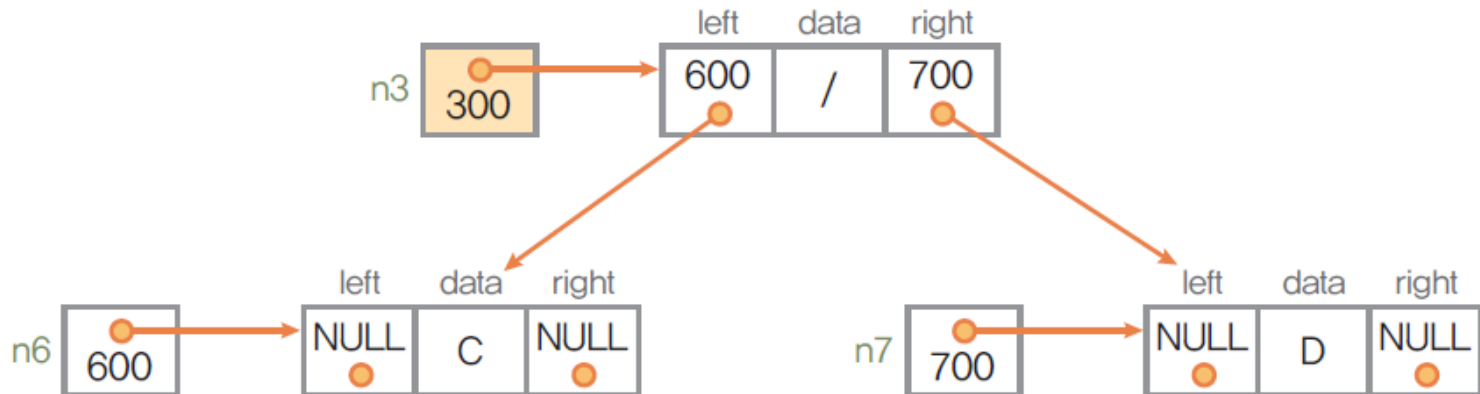
- 51행 : 데이터 필드가 B' '이고 왼쪽 서브 트리와 오른쪽 서브 트리가 공백 노드인 이진 트리 노드 n5 를 만듦



- 52행 : 데이터 필드가 A' '이고 왼쪽 서브 트리와 오른쪽 서브 트리가 공백 노드인 이진 트리 노드 n4 를 만듦

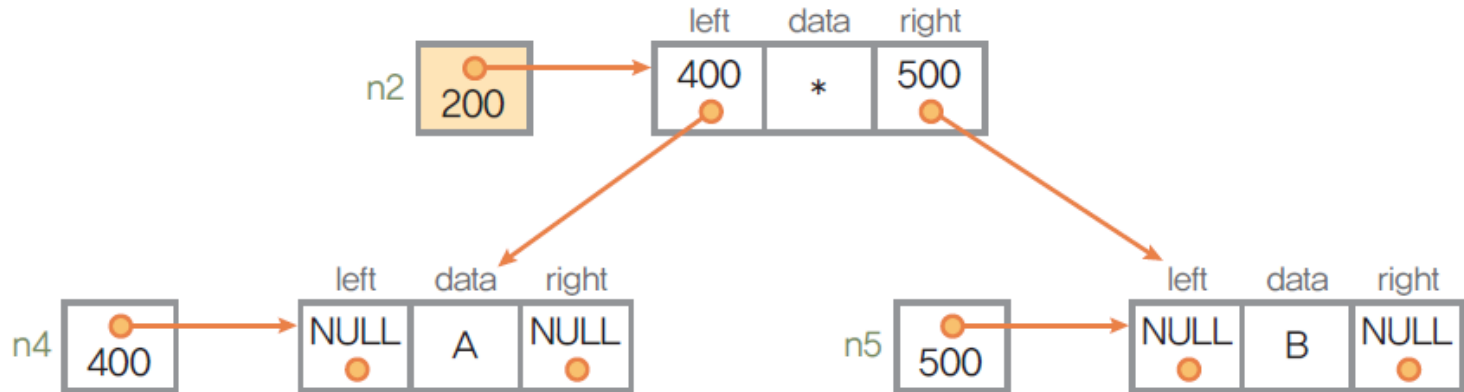


- 53행 : 데이터 필드가 '/'이고, 왼쪽 링크 필드에 노드 n6, 오른쪽 링크 필드에 노드 n7이 연결된 이진 트리 노드 n3을 만듦

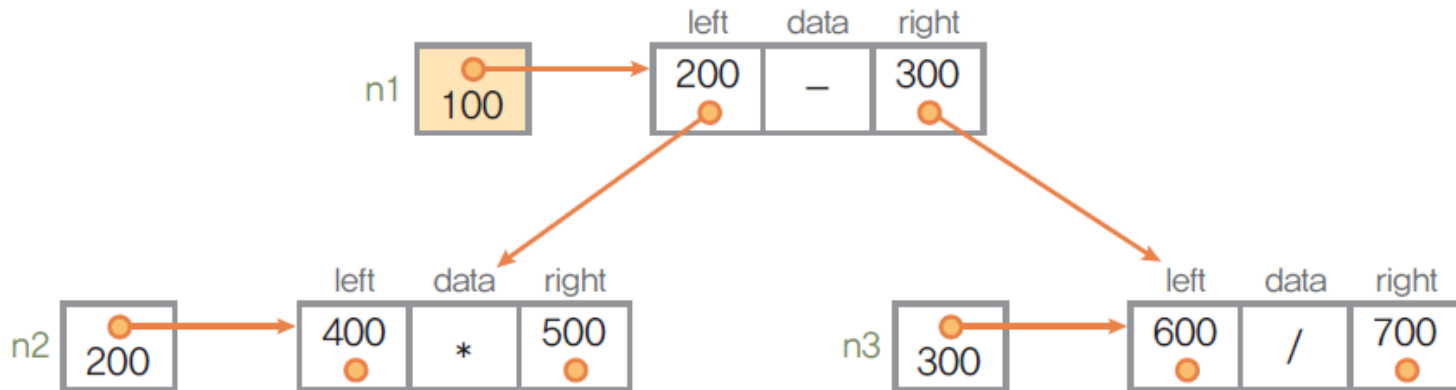


4. 이진 트리의 순회

- 54행 : 데이터 필드가 '*'이고 왼쪽 링크 필드에 노드 n4, 오른쪽 링크 필드에 노드 n5가 연결된 이진 트리 노드 n2를 만들



- 55행 : 데이터 필드가 '-'이고 왼쪽 링크 필드에 노드 n2, 오른쪽 링크 필드에 노드 n3이 연결된 이진 트리 노드 n1을 만들



4. 이진 트리의 순회

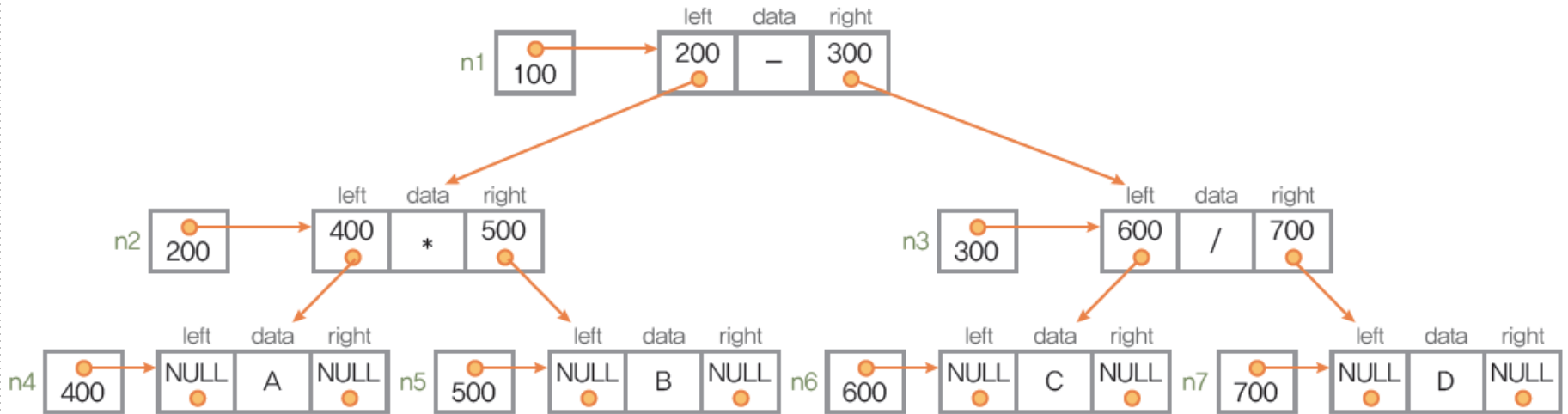


그림 7-27 makeRootNode()를 이용한 이진 트리의 구성 과정



4. 이진 트리의 순회

❖ 이진 트리 순회의 응용 프로그램

- 컴퓨터의 폴더 구조가 이진 트리 구조인 경우 각 폴더를 순회하면서 용량을 계산하면 내 컴퓨터의 전체 용량이 계산 가능함.
 - 폴더 전체 용량은 폴더에 저장된 파일 용량과 하위 폴더의 용량을 합한 값.
상위 폴더 용량을 계산하려면 먼저 하위 폴더 용량을 계산해야 하므로 후위 순회를 사용



4. 이진 트리의 순회

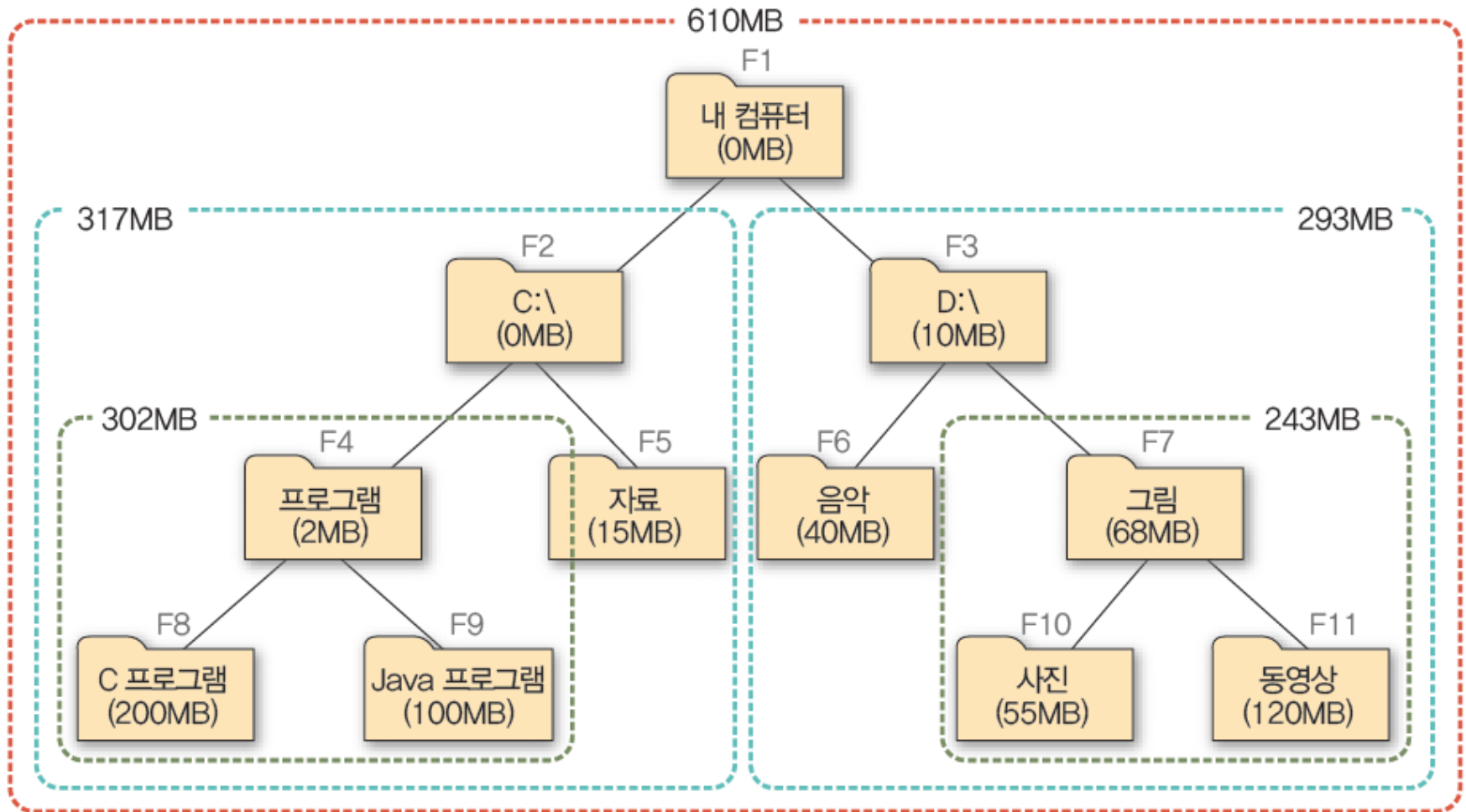


그림 7-28 후위 순회의 사용 예 : 컴퓨터 폴더 구조에서 전체 용량 계산하기

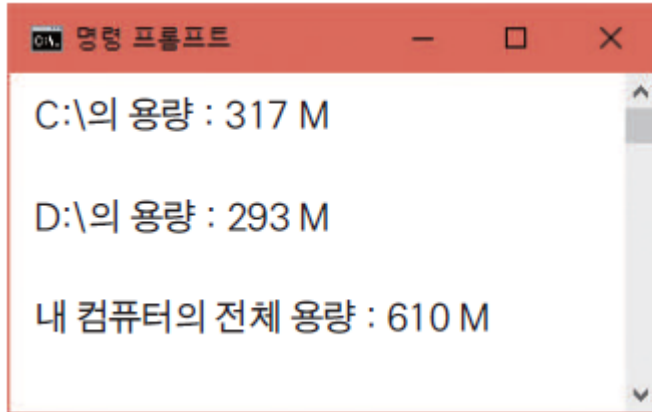
4. 이진 트리의 순회

- [프로그램] 전체 용량 = [프로그램] 용량 2MB + 하위 폴더 용량([C 프로그램] 용량 200MB+ [Java 프로그램] 용량 100MB) = 2MB+(200MB+100MB) = 302MB
- [C:\W] 전체 용량 = [C:\W] 용량 0MB + 하위 폴더 용량([프로그램] 전체 용량 302MB + [자료] 용량 15MB)= 0MB+(302MB+15MB) = 317MB
- [그림] 전체 용량 = [그림] 용량 68MB + 하위 폴더 용량([사진] 용량 55MB + [동영상] 용량 120MB) =68MB+(55MB+120MB) = 243MB
- [D:\W] 전체 용량 = [D:\W] 용량 10MB+ 하위 폴더 용량([음악] 용량 40MB + [그림] 전체 용량 243MB) =10MB+(40MB+243MB) = 293MB
- [내 컴퓨터] 전체 용량 = [내 컴퓨터] 용량 0MB + 하위 폴더 용량([C:\W] 전체 용량 317MB + [D:\W] 전체 용량 293MB) = 0MB+(317MB+293MB) = 610MB



4. 이진 트리의 순회

- 이진 트리의 후위 순회를 이용해 폴더 용량 계산 프로그램 : [교재 340p](#)
- 실행 결과



```
명령 프롬프트
C:\의 용량 : 317 M
D:\의 용량 : 293 M
내 컴퓨터의 전체 용량 : 610 M
```



4. 이진 트리의 순회

❖ 스레드 이진 트리 Thread Binary Tree

- 재귀호출 없이 순회할 수 있도록 수정한 이진 트리
 - 재귀호출 방식은 알고리즘이나 함수 구현은 간단하지만, 수행 성능 측면에서는 시스템 스택을 사용하면서 호출과 복귀를 관리해야 하고 이진 트리의 하위 레벨로 내려갈수록 재귀호출 깊이가 깊어지므로 비효율적임.
- 스레드(Thread)
 - 스레드 이진 트리에서 자식 노드가 없는 경우 링크 필드를 널 포인터 대신 순회 순서상의 다른 노드를 가리키도록 설정. 이런 링크 필드를 스레드라 함



4. 이진 트리의 순회

■ 스레드 이진 트리 노드

- 이진 트리 노드 구조에 isThread 필드를 추가하여 정의
- isThread 필드는 링크 필드가 자식 노드에 대한 포인터인지 아니면 자식 노드 대신 스레드가 저장되어 있는지 구별하기 위한 태그 필드

```
typedef struct treeNode {  
    char data;  
    struct treeNode *left;  
    struct treeNode *right;  
    int isThreadLeft;  
    int isThreadRight;  
} treeNode;
```

그림 7-29 스레드 이진 트리의 노드 정의



4. 이진 트리의 순회

- $A * B - C / D$ 수식의 이진 트리에서 중위 순회를 한다면, 왼쪽 스레드에는 선행자를 설정하고 오른쪽 스레드에는 후속자를 설정. 중위 순회 경로가 $A \rightarrow * \rightarrow B \rightarrow - \rightarrow C \rightarrow / \rightarrow D$ 이므로 B의 선행자는 '*'이고 B의 후속자는 '-'임

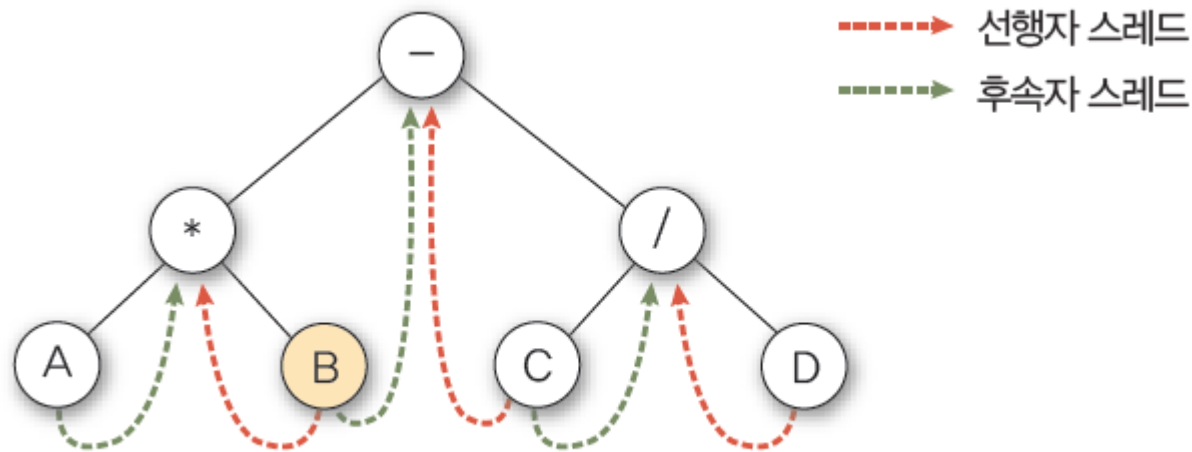


그림 7-30 선행자/후속자 스레드를 갖는 스레드 이진 트리



4. 이진 트리의 순회

- 순회를 위해서 후속자 정보만 필요한 경우에는 오른쪽 링크 필드만 스레드로 사용하는 스레드 이진 트리를 사용할 수 있음

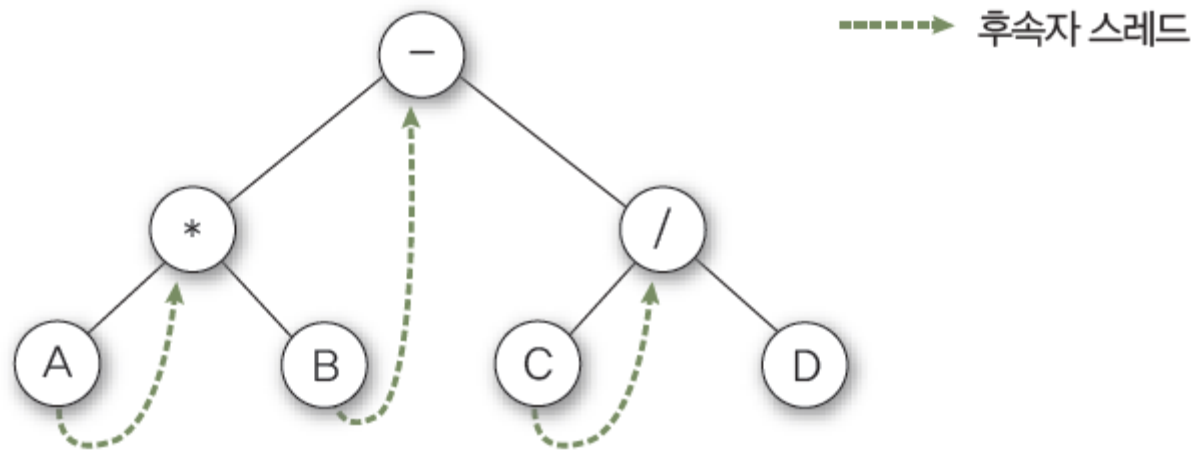
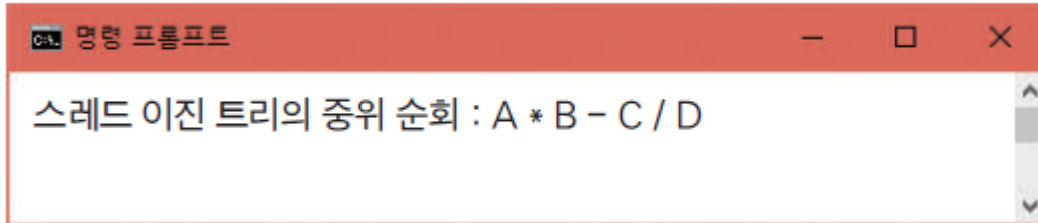


그림 7-31 후속자 스레드를 갖는 스레드 이진 트리



4. 이진 트리의 순회

- 스레드 이진 트리 구현하기 프로그램 : [교재 343p](#)
- 실행 결과



```
명령 프롬프트
스레드 이진 트리의 중위 순회 : A * B - C / D
```



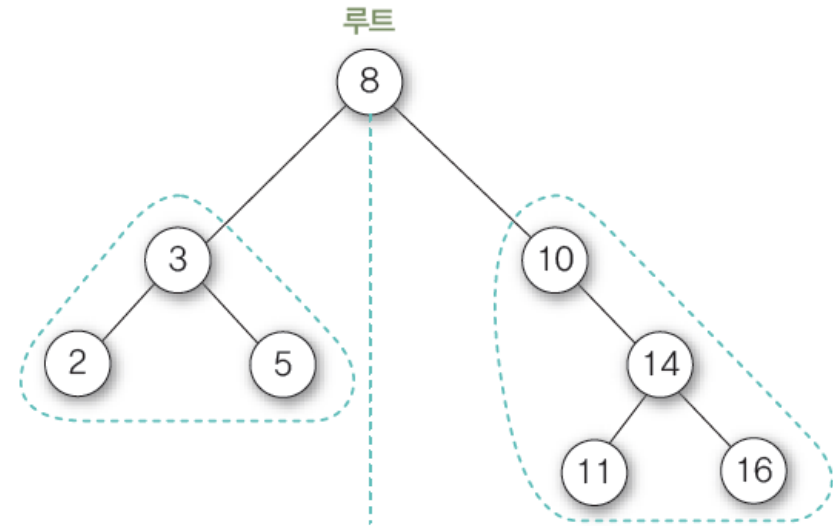
5. 이진 탐색 트리

❖ 이진 탐색 트리 binary search tree

- 이진 트리를 탐색용 자료구조로 사용하기 위해 원소 크기에 따라 노드 위치를 정의한 것

- 모든 원소는 서로 다른 유일한 키를 갖는다.
- 왼쪽 서브 트리에 있는 원소들의 키는 그 루트의 키보다 작다.
- 오른쪽 서브 트리에 있는 원소들의 키는 그 루트의 키보다 크다.
- 왼쪽 서브 트리과 오른쪽 서브 트리도 이진 탐색 트리이다.

그림 7-32 이진 탐색 트리의 정의



왼쪽 서브 트리의 키값 < 루트의 키값 < 오른쪽 서브 트리의 키값

그림 7-33 이진 탐색 트리의 구조



❖ 이진 탐색 트리의 탐색 연산

- 루트에서 시작
- 탐색할 키값 x 를 루트 노드의 키값과 비교
 - (키값 $x =$ 루트 노드의 키값)인 경우 : 원하는 원소를 찾았으므로 탐색연산 성공
 - (키값 $x <$ 루트 노드의 키값)인 경우 : 루트노드의 왼쪽 서브트리에 대해서 탐색연산 수행
 - (키값 $x >$ 루트 노드의 키값)인 경우 : 루트노드의 오른쪽 서브트리에 대해서 탐색연산 수행
- 서브트리에 대해서 순환적으로 탐색 연산을 반복



5. 이진 탐색 트리

알고리즘 7-4 이진 탐색 트리의 노드 탐색

```
searchBST(bsT, x)
  p ← bsT;
  if (p = NULL) then
    return NULL;
  if (x = p.key) then
    return p;
  if (x < p.key) then
    return searchBST(p.left, x);
  else return searchBST(p.right, x);
end searchBST()
```



5. 이진 탐색 트리

- 이진 탐색 트리에서 원소 11을 탐색

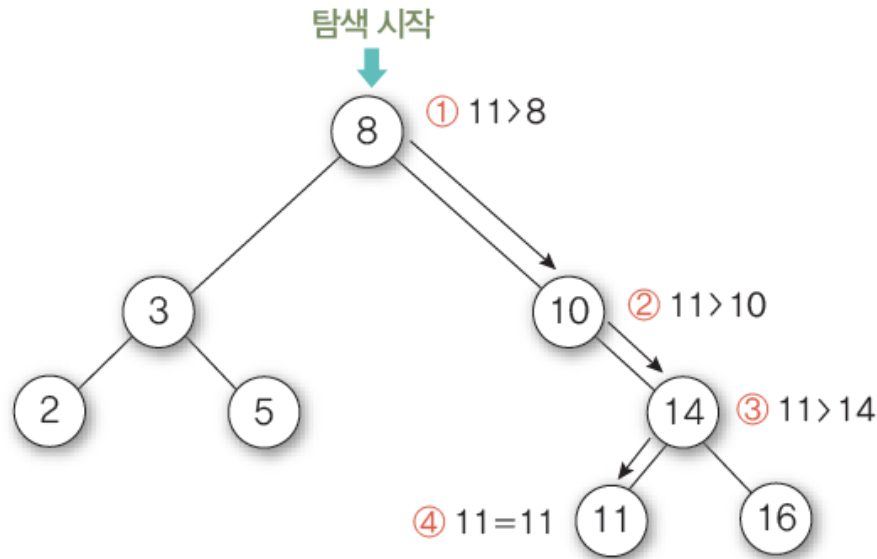


그림 7-34 이진 탐색 트리에서의 탐색 과정 예

- ① (찾는 키값 11 > 루트 노드의 키값 8)이므로 오른쪽 서브 트리 탐색
- ② (찾는 키값 11 > 노드의 키값 10)이므로 다시 오른쪽 서브 트리 탐색
- ③ (찾는 키값 11 < 노드의 키값 14)이므로 왼쪽 서브 트리 탐색
- ④ (찾는 키값 11 = 노드의 키값 11)이므로 탐색 성공으로 연산 종료



❖ 이진 탐색 트리의 삽입 연산

1) 먼저 탐색 연산을 수행

- 삽입할 원소와 같은 원소가 트리에 있으면 삽입할 수 없으므로, 같은 원소가 트리에 있는지 탐색하여 확인
- 탐색에서 탐색 실패가 결정되는 위치가 삽입 위치가 됨

2) 탐색 실패한 위치에 원소를 삽입



5. 이진 탐색 트리

- 이진 탐색 트리에서 삽입 연산을 하는 알고리즘
 - 삽입할 자리를 찾기 위해 포인터 p를 사용, 삽입할 노드의 부모 노드를 지정하기 위해 포인터q 를 사용

알고리즘 7-5 이진 탐색 트리의 노드 삽입

```
insertBST(bsT, x)
```

```
  p ← bsT
```

```
  while (p ≠ NULL) do {
```

```
    if (x = p.key) then return;
```

```
    q ← p;
```

```
    if (x < p.key) then p ← p.left;
```

```
    else p ← p.right;
```

```
  }
```

① 삽입할 노드 탐색

```
  new ← getNode();
```

```
  new.key ← x;
```

```
  new.left ← NULL;
```

```
  new.right ← NULL;
```

② 삽입할 노드 생성



5. 이진 탐색 트리

```
if (bsT = NULL) then bsT ← new;  
else if (x < q.key) then q.left ← new;  
else q.right ← new;  
return;  
end insertBST()
```

} ③ 삽입 노드 연결



5. 이진 탐색 트리

■ 이진 탐색 트리에 원소 4를 삽입 하기

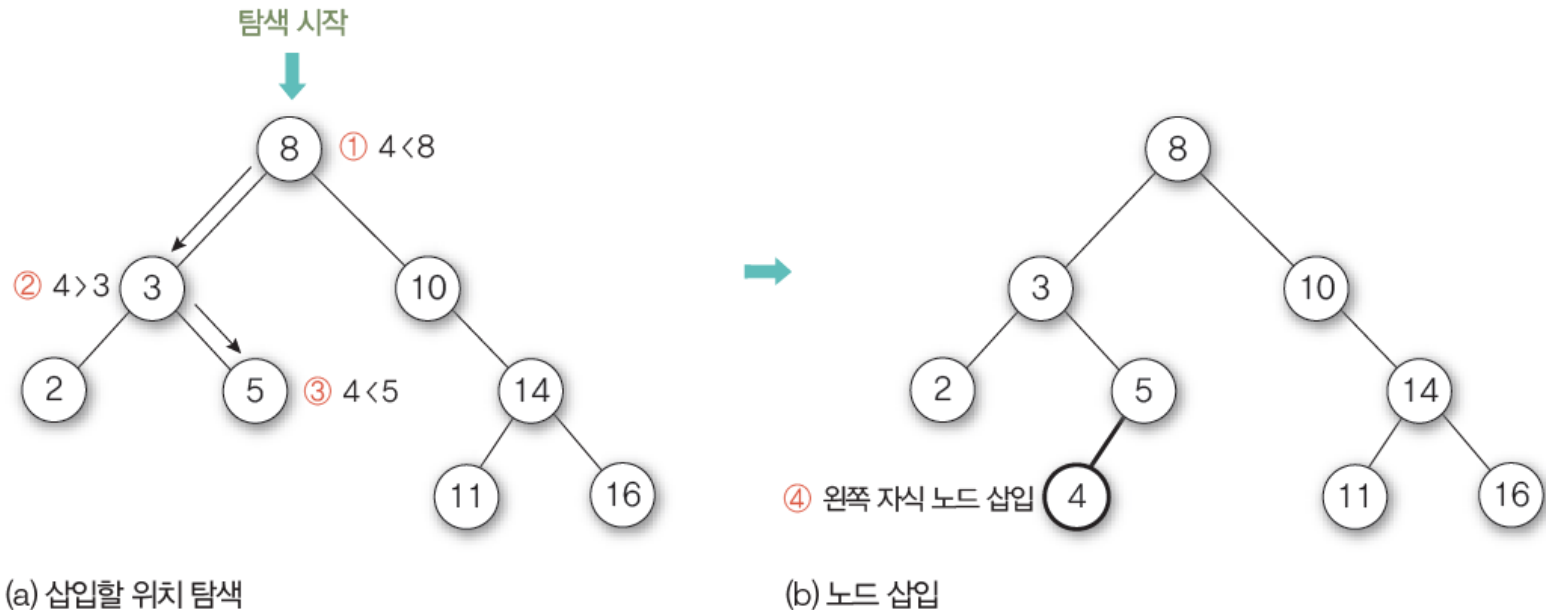


그림 7-35 이진 탐색 트리에서의 삽입 과정 예

- ① (찾는 키값 $4 <$ 루트 노드의 키값 8)이므로 왼쪽 서브 트리를 탐색
- ② (찾는 키값 $4 >$ 노드의 키값 3)이므로 오른쪽 서브 트리를 탐색
- ③ (찾는 키값 $4 <$ 노드의 키값 5)이므로 왼쪽 서브 트리를 탐색해야 하지만, 왼쪽 자식 노드가 없으므로 노드 5의 왼쪽 자식 노드에서 탐색 실패가 발생
- ④ 실패가 발생한 자리, 즉 노드 5의 왼쪽 자식 노드 자리에 노드 4를 삽입

5. 이진 탐색 트리

- 이진 탐색 트리에 원소 4를 삽입하는 과정을 연결 자료구조로 표현

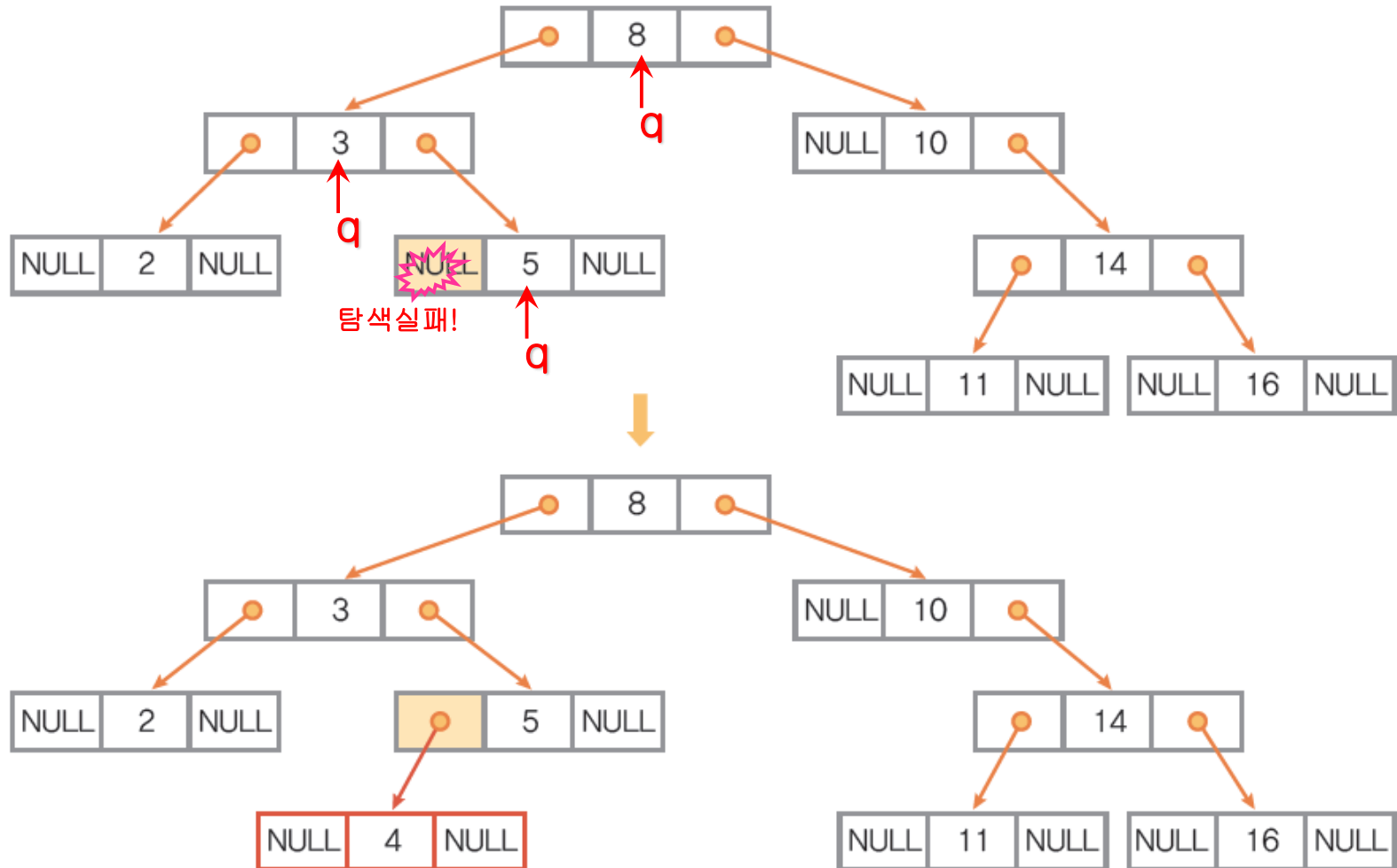


그림 7-36 이진 탐색 트리에 원소 4를 삽입하기 전과 후로 나눠 연결 자료구조로 표현



❖ 이진 탐색 트리의 삭제 연산

1) 먼저 탐색 연산을 수행

- 삭제할 노드의 위치를 알아야 하므로 트리를 탐색

2) 탐색하여 찾은 노드를 삭제

- 노드의 삭제 후에도 이진 탐색 트리를 유지해야 하므로 삭제 노드의 경우에 대한 후속 처리(이진 탐색 트리의 재구성 작업)가 필요함
 - 삭제할 노드의 경우
 - 삭제할 노드가 단말 노드인 경우(차수 = 0)
 - 삭제할 노드가 자식 노드를 한 개 가진 경우(차수 = 1)
 - 삭제할 노드가 자식 노드를 두 개 가진 경우(차수 = 2)



5. 이진 탐색 트리

- 삭제할 노드가 단말 노드인 경우(차수 = 0)의 삭제 연산
 - 노드 4를 삭제하는 경우

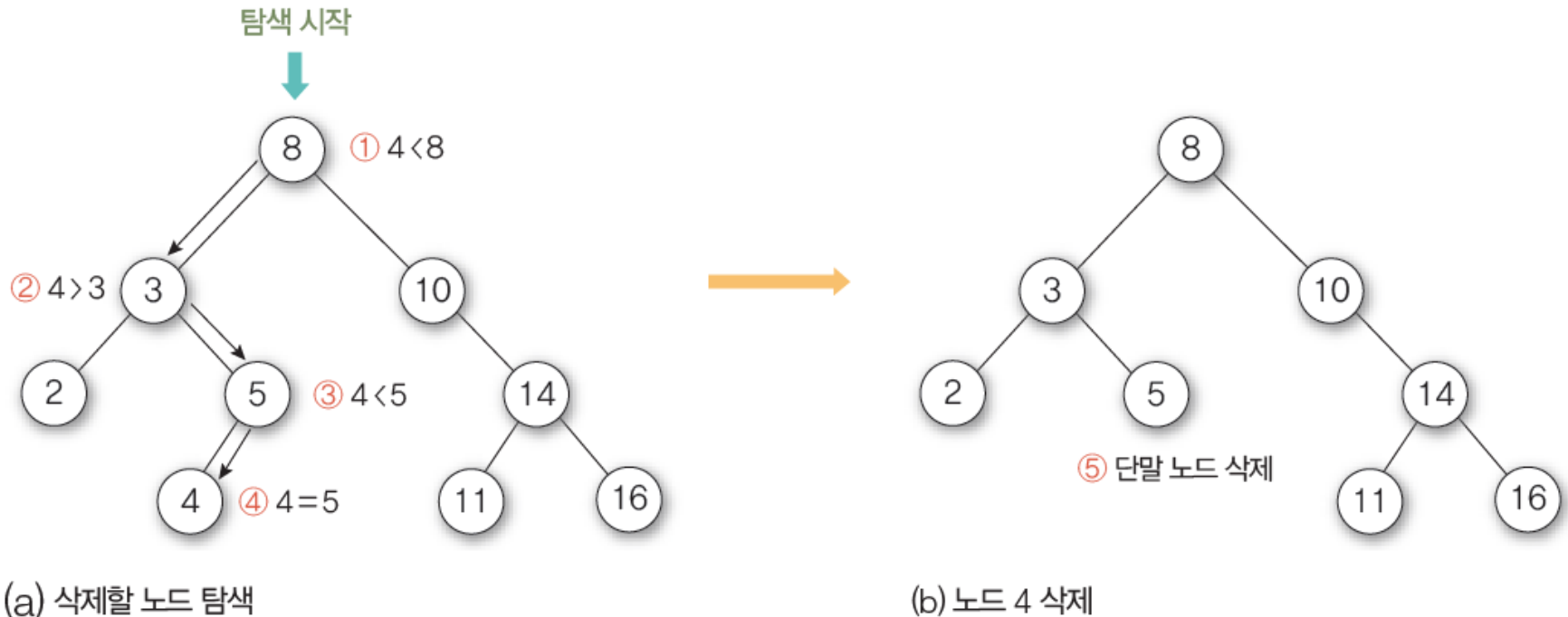


그림 7-37 이진 탐색 트리에서 단말 노드 4를 삭제하는 예



5. 이진 탐색 트리

- 4를 삭제하기 전과 후로 연결 자료구조를 표현

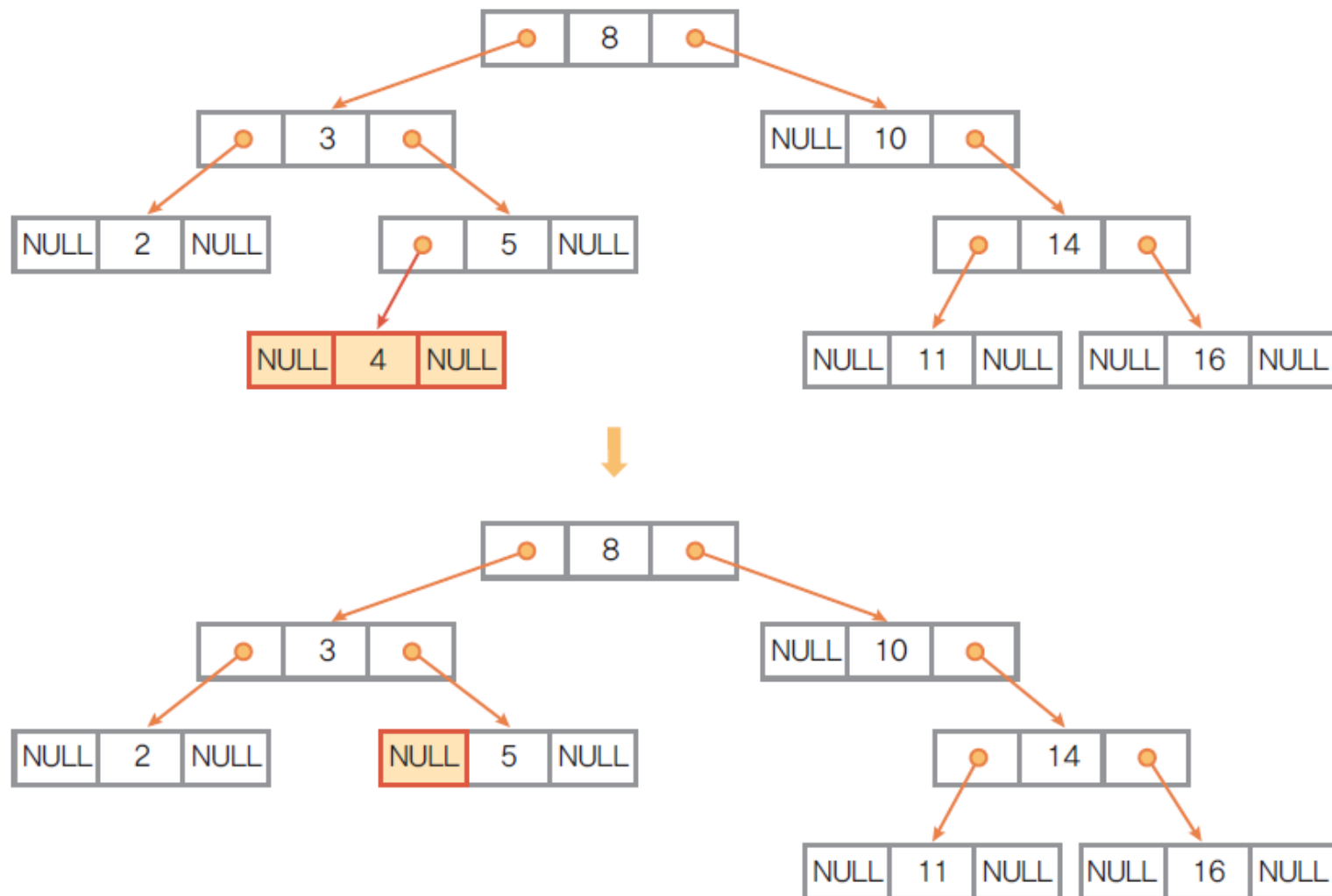
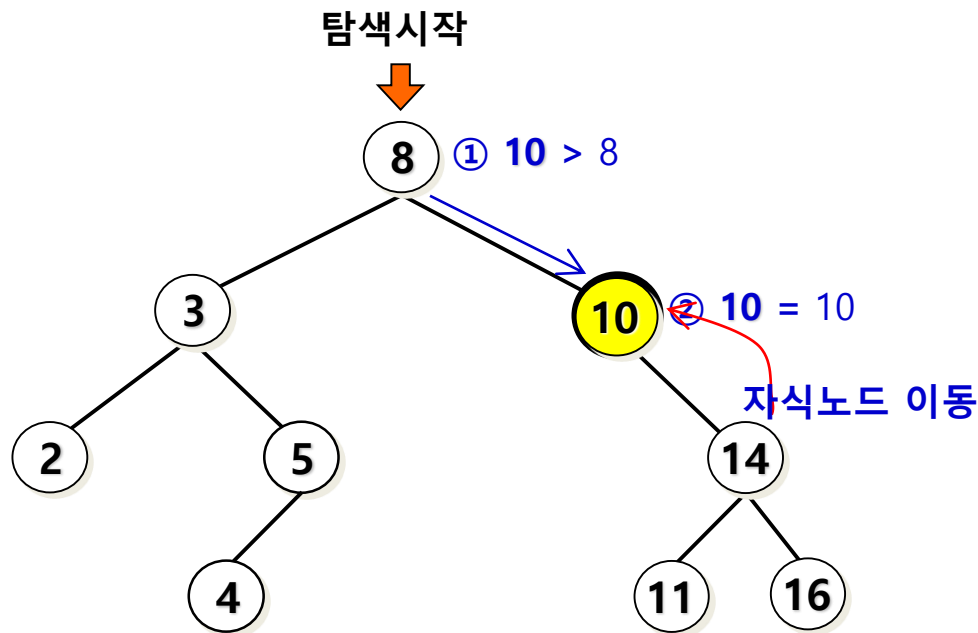


그림 7-38 이진 탐색 트리에서 단말 노드 4를 삭제하기 전과 후로 나눠 연결 자료구조로 표현

5. 이진 탐색 트리

- 삭제할 노드가 자식 노드를 한 개 가진 경우(차수 = 1)의 삭제 연산
 - 노드를 삭제하면, 자식 노드는 트리에서 연결이 끊어져서 고아가 됨
 - 후속 처리 : 삭제한 부모노드의 자리를 자식노드에게 물려줌
 - 노드 10을 삭제하는 경우



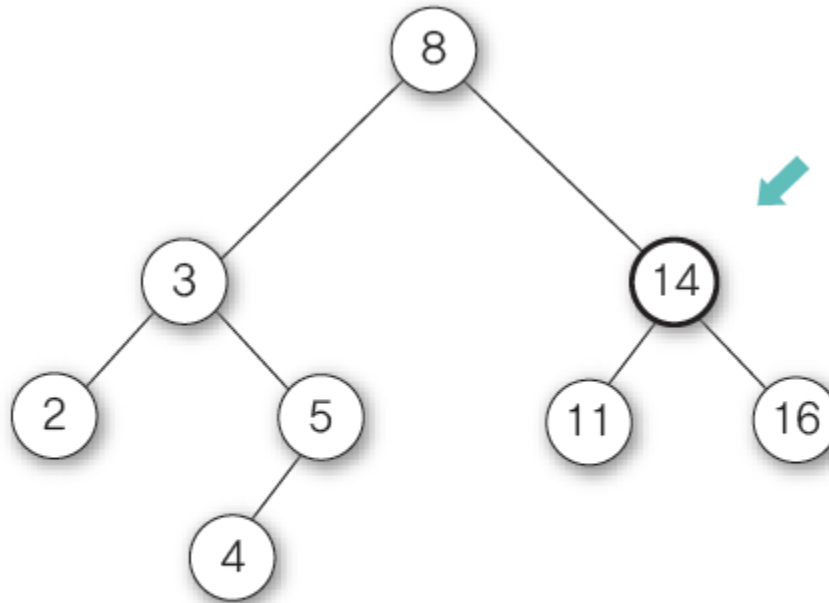
1단계: 삭제할 노드 **탐색**

2단계: 탐색한 노드 **삭제**

3단계: **후속처리**



5. 이진 탐색 트리



(c) 자식 노드의 위치 조정

그림 7-39 이진 탐색 트리에서 자식 노드가 하나인 노드 10을 삭제하는 예



5. 이진 탐색 트리

- 노드 10을 삭제하는 경우에 대한 단순 연결 리스트 표현

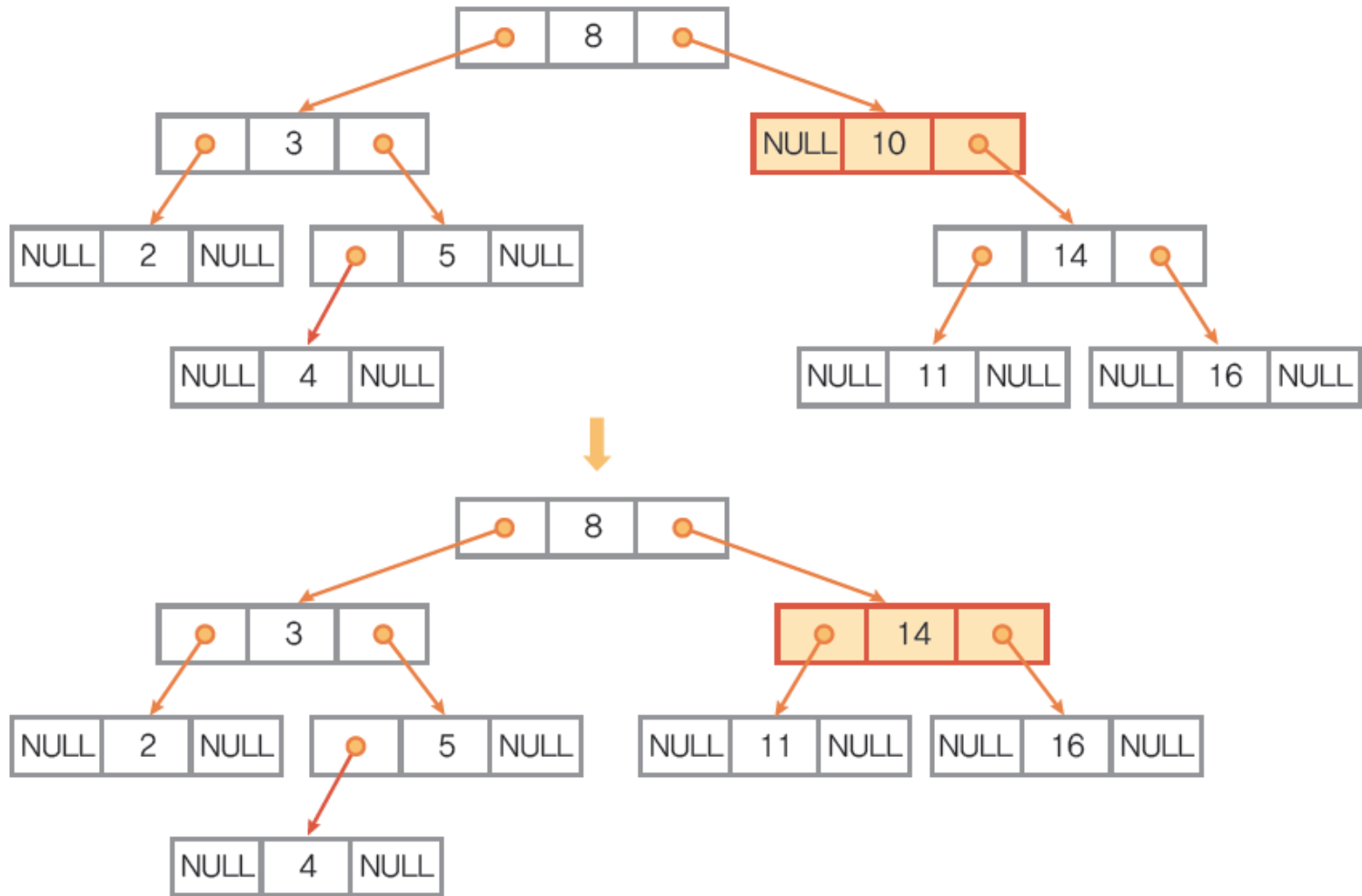


그림 7-40 이진 탐색 트리에서 자식 노드가 하나인 노드 10을 삭제하기 전과 후로 나눠 연결 자료구조로 표현



5. 이진 탐색 트리

- 삭제할 노드가 자식 노드를 두 개 가진 경우(차수 = 2)의 삭제 연산
 - 노드를 삭제하면, 자식 노드들은 트리에서 연결이 끊어져서 고아가 됨
 - 후속 처리 : 삭제한 노드의 자리를 자손 노드들 중에서 선택한 후계자에게 물려줌
- 후계자 선택 방법
 - 왼쪽 서브트리에서 가장 큰 자손노드 선택 : 왼쪽 서브트리의 오른쪽 링크를 따라 계속 이동하여 오른쪽 링크 필드가 NULL인 노드 즉, 가장 오른쪽 노드가 후계자가 됨
 - 오른쪽 서브트리에서 가장 작은 자손노드 선택 : 오른쪽 서브트리에서 왼쪽 링크를 따라 계속 이동하여 왼쪽 링크 필드가 NULL인 노드 즉, 가장 왼쪽에 있는 노드가 후계자가 됨



5. 이진 탐색 트리

- 삭제한 노드의 자리를 물려받을 수 있는 자손 노드

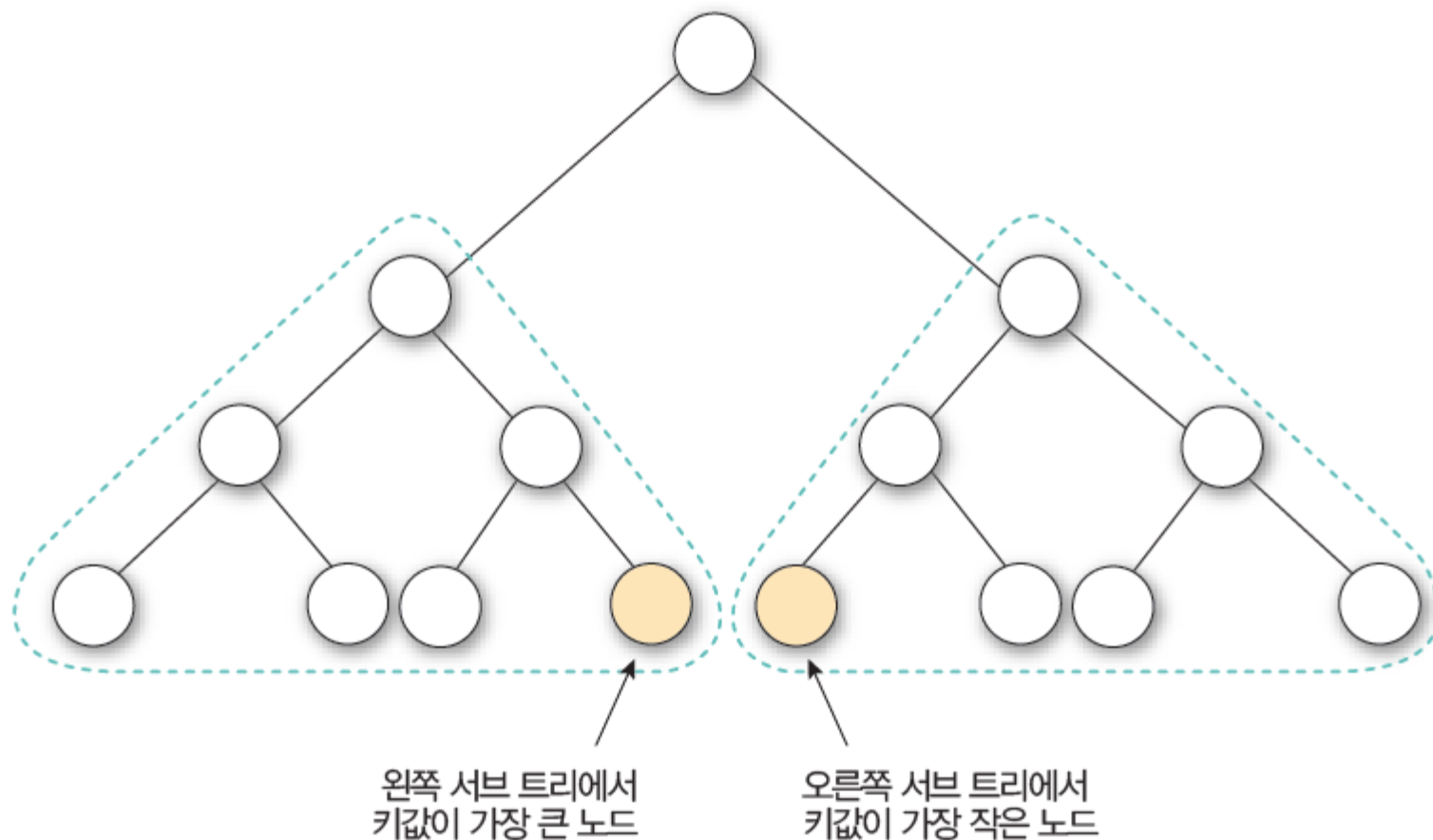
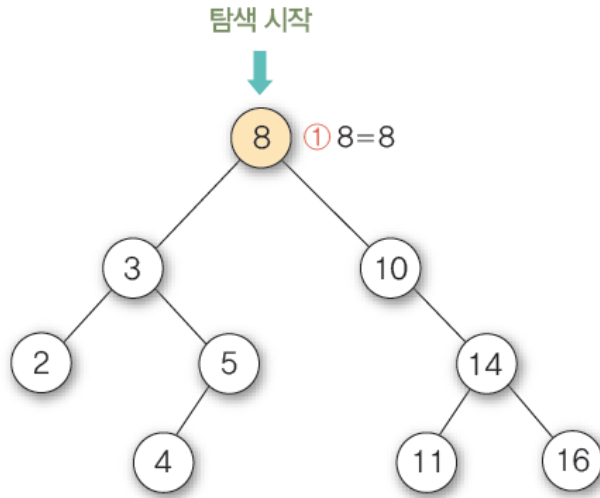


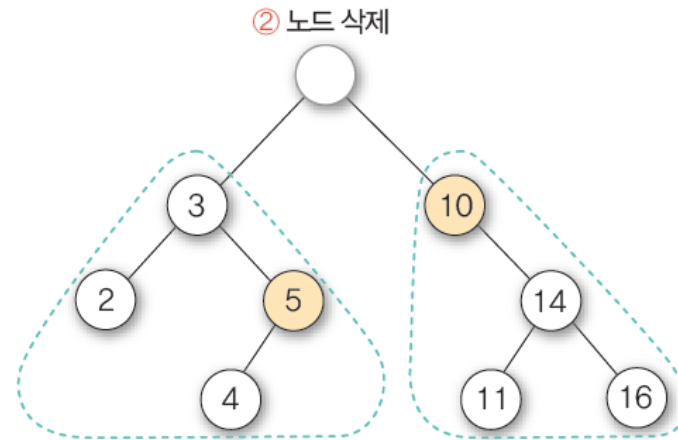
그림 7-41 삭제할 노드의 자리를 물려받을 수 있는 자손 노드

5. 이진 탐색 트리

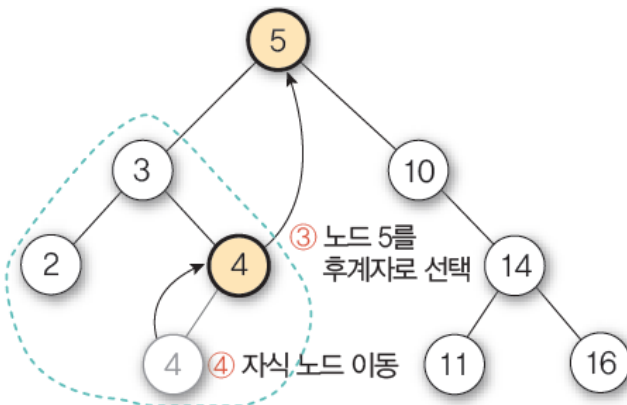
- 노드 8을 삭제하는 경우



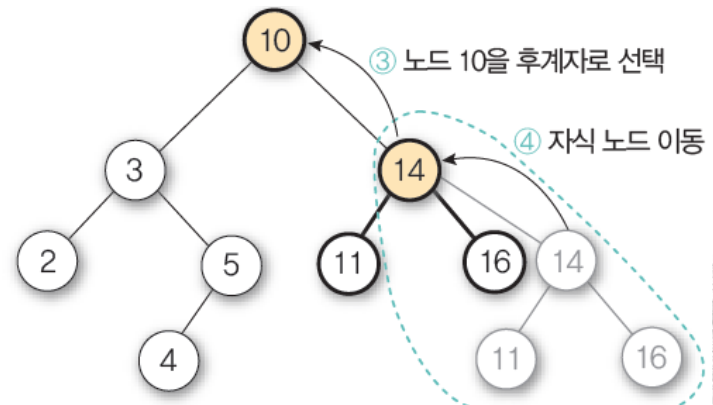
(a) 삭제할 노드 탐색



(b) 노드 삭제



(c)-1 노드 5를 후계자로 선택하는 경우



(c)-2 노드 10을 후계자로 선택하는 경우

그림 7-42 이진 탐색 트리에서 자식 노드가 둘인 노드 8을 삭제하는 예



5. 이진 탐색 트리

- 노드 5를 후계자로 선택한 경우
 - ① 후계자 노드 5를 원래자리에서 삭제하여, 삭제노드 8의 자리를 물려줌
 - ② 후계자 노드 5의 원래자리는 자식노드 4에게 물려주어 이진 탐색 트리를 재구성 (자식노드가 하나인 노드 삭제 연산의 후속처리 수행)
- 노드 10을 후계자로 선택한 경우
 - ① 후계자 노드 10을 원래자리에서 삭제하여, 삭제노드 8의 자리를 물려줌
 - ② 후계자 노드 10의 원래자리는 자식노드 14에게 물려주어 이진 탐색 트리를 재구성 (자식노드가 하나인 노드 삭제 연산의 후속처리 수행)



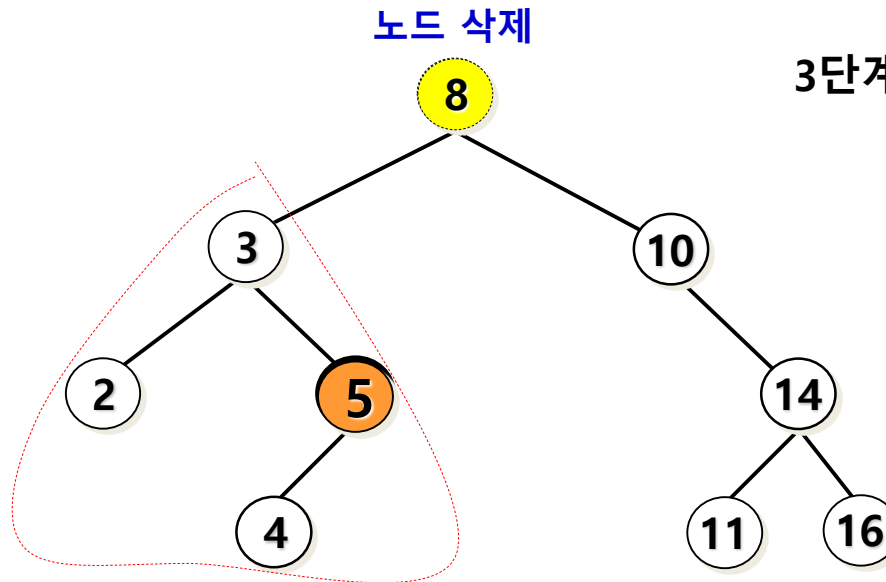
5. 이진 탐색 트리

- 노드 5를 후계자로 선택한 경우

1단계: 노드 삭제

2단계: 삭제한 노드의 자리를
후계자에게 물려주기

3단계: 후계자노드의 원래자리를
자식노드에게 물려주기



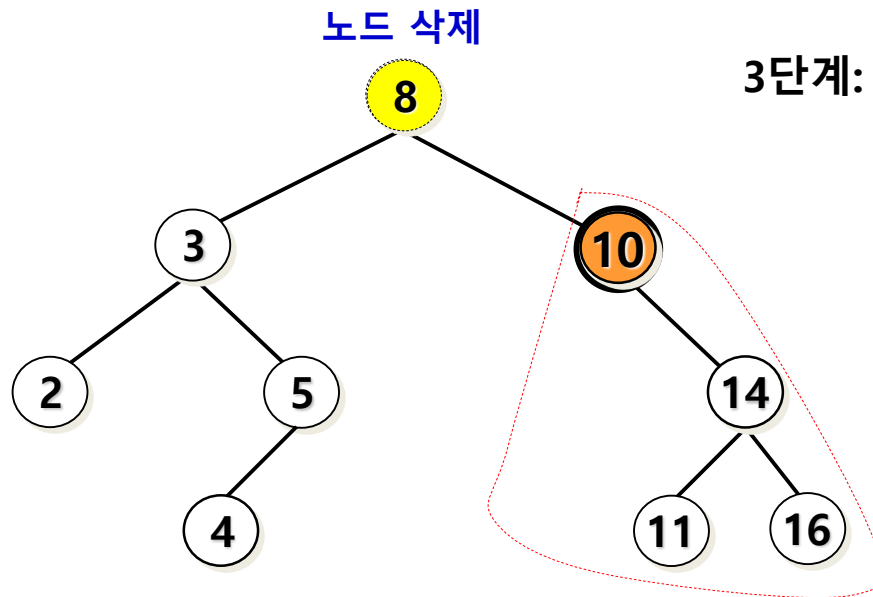
5. 이진 탐색 트리

- 노드 10을 후계자로 선택한 경우

1단계: 노드 삭제

2단계: 삭제한 노드의 자리를
후계자에게 물려주기

3단계: 후계자노드의 원래자리를
자식노드에게 물려주기



5. 이진 탐색 트리

■ 이진 탐색 트리의 노드 삭제

알고리즘 7-6 이진 탐색 트리의 노드 삭제

```
deleteBST(bsT, x)
  p ← 삭제할 노드;
  parent ← 삭제할 노드의 부모 노드;

  // 삭제할 노드가 없는 경우
  if (p = NULL) then return;

  // 삭제할 노드의 차수가 0인 경우
  if (p.left = NULL and p.right = NULL) then {
    if (parent.left = p) then parent.left ← NULL;
    else parent.right ← NULL;
  }
```



5. 이진 탐색 트리

// 삭제할 노드의 차수가 1인 경우

```
else if (p.left = NULL or p.right = NULL) then {  
    if (p.left ≠ NULL) then{  
        if (parent.left = p) then parent.left ← p.left;  
        else parent.right ← p.left;  
    }  
    else {  
        if (parent.left = p) then parent.left ← p.right;  
        else parent.right ← p.right;  
    }  
}
```

// 삭제할 노드의 차수가 2인 경우

```
else if (p.left ≠ NULL and p.right ≠ NULL) then {  
    q ← maxNode(p.left);    // 왼쪽 서브 트리에서 후계자 노드를 포인터 q로 지정  
    p.key ← q.key;  
    deleteBST(p.left, p.key); // 후계자 노드 자리에 대한 2차 재구성  
}
```

```
end deleteBST()
```



5. 이진 탐색 트리

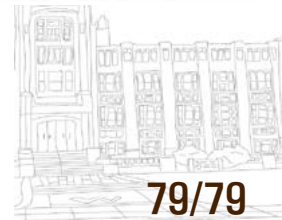
- 이진 탐색 트리의 연산 프로그램 : [교재 356p](#)
- 실행 결과

```
영웅 프로그램
*-----*
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
*-----*
메뉴 입력 >> 1
[이진 트리 출력] A_B_D_E_G_H_I_J_M_N_O_
*-----*
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
*-----*
메뉴 입력 >> 2
삽입할 문자를 입력하세요 : C
```

```
영웅 프로그램
*-----*
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
*-----*
메뉴 입력 >> 1
[이진 트리 출력] A_B_C_D_E_G_H_I_J_M_N_O_
*-----*
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
*-----*
메뉴 입력 >> 3
삭제할 문자를 입력하세요 : A
*-----*
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
*-----*
```

```
메뉴 입력 >> 1
[이진 트리 출력] B_C_D_E_G_H_I_J_M_N_O_
*-----*
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
*-----*
메뉴 입력 >> 4
찾을 문자를 입력하세요 : A

찾는 키가 없습니다!
문자를 찾지 못했습니다.
*-----*
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
*-----*
메뉴 입력 >>
```



Thank You

