

이진트리

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

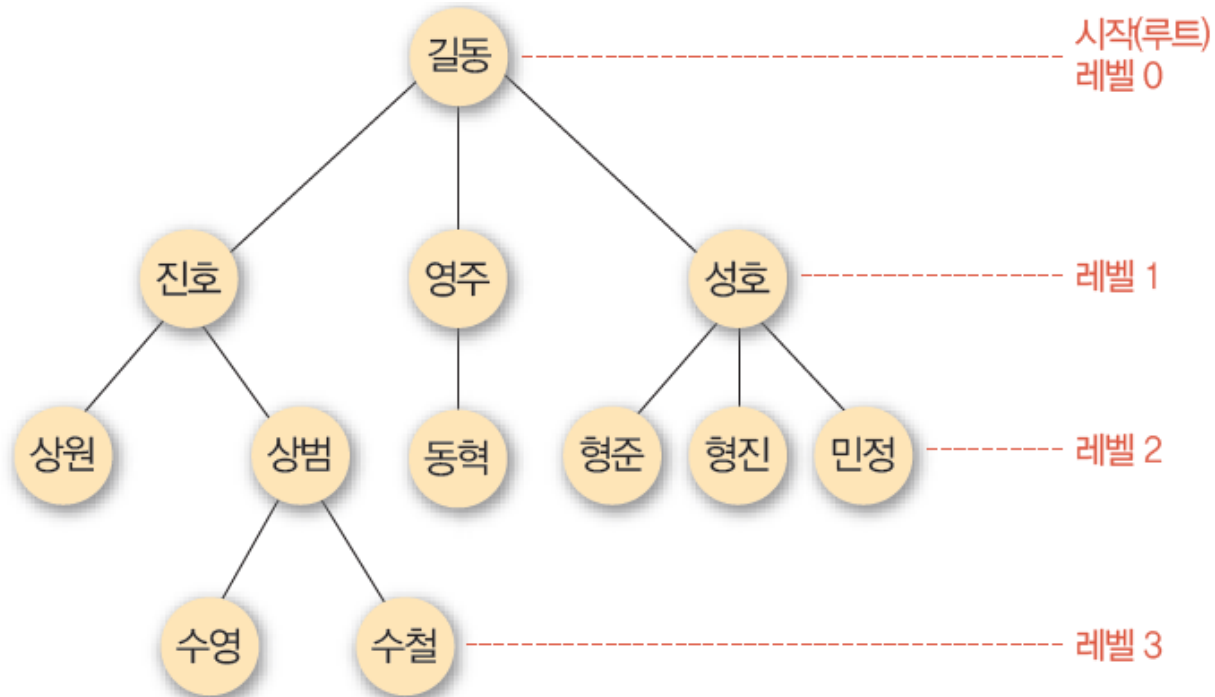


그림 7-1 트리 자료구조의 예 : 가계도

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

1. 트리의 이해

■ 트리 A

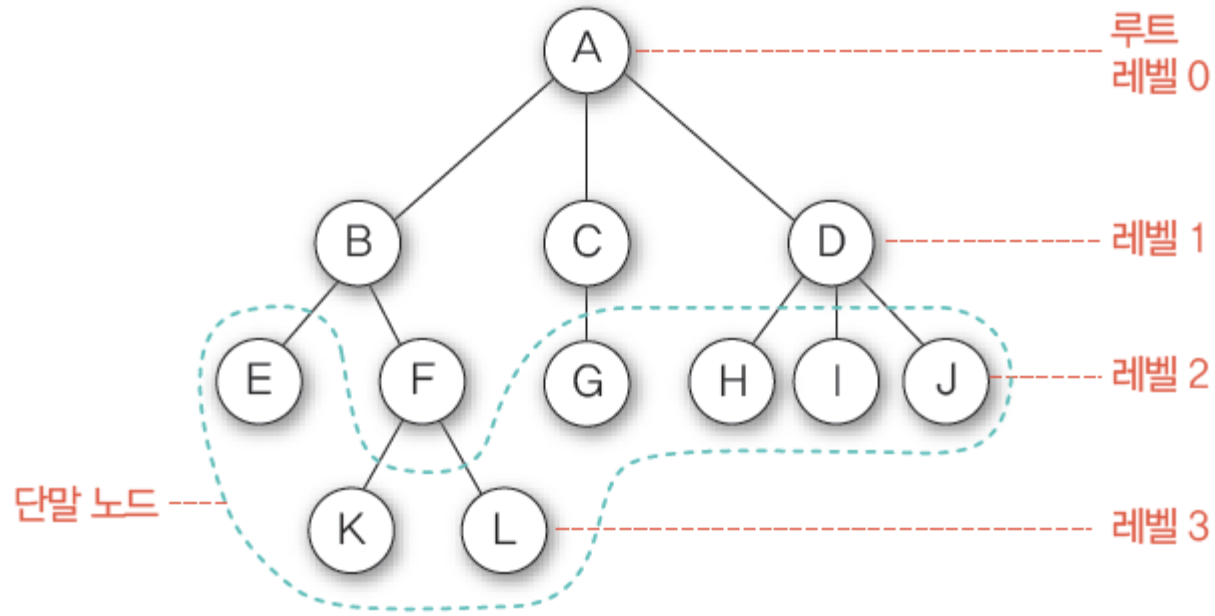


그림 7-2 트리의 구조와 구성 요소

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

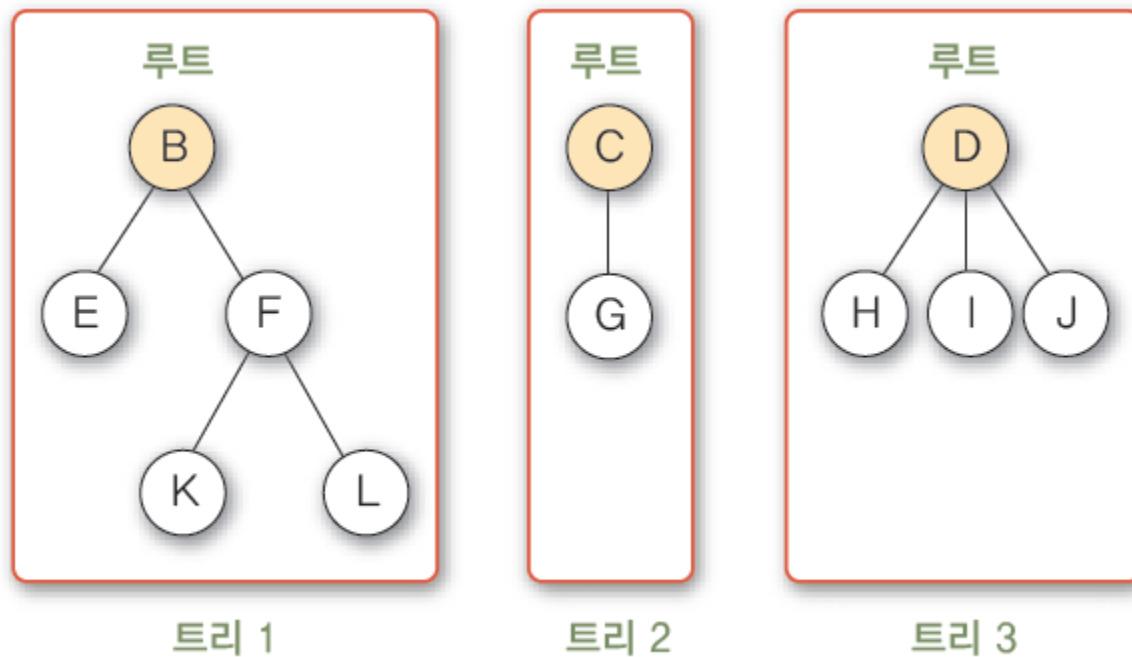


그림 7-3 [그림 7-2]에서 루트 노드 A를 제거하여 만든 포리스트

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
 - $0 \leq \text{노드의 차수} \leq 2$

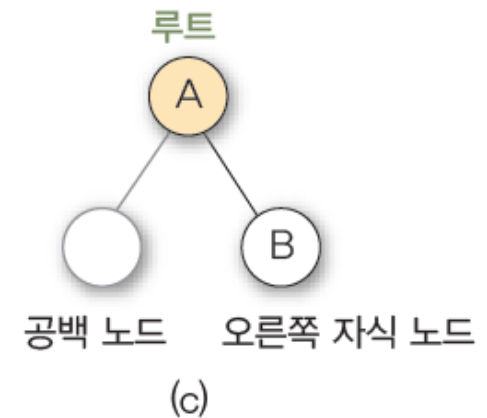
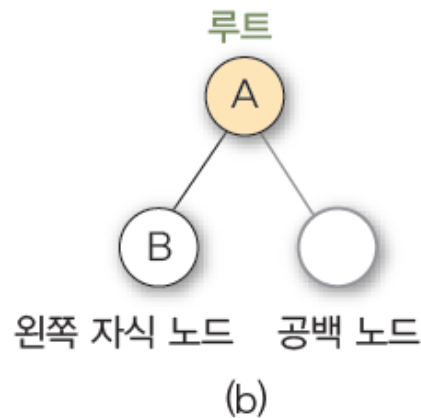
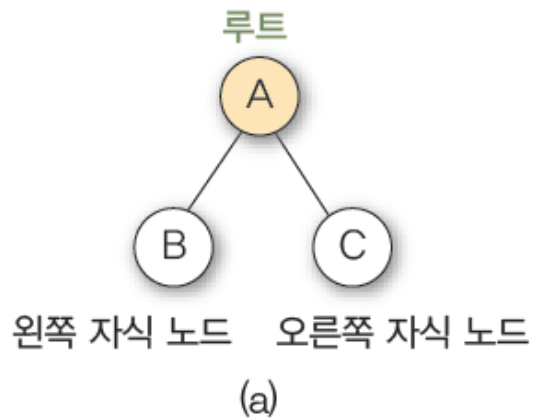


그림 7-4 이진 트리의 기본 구조

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

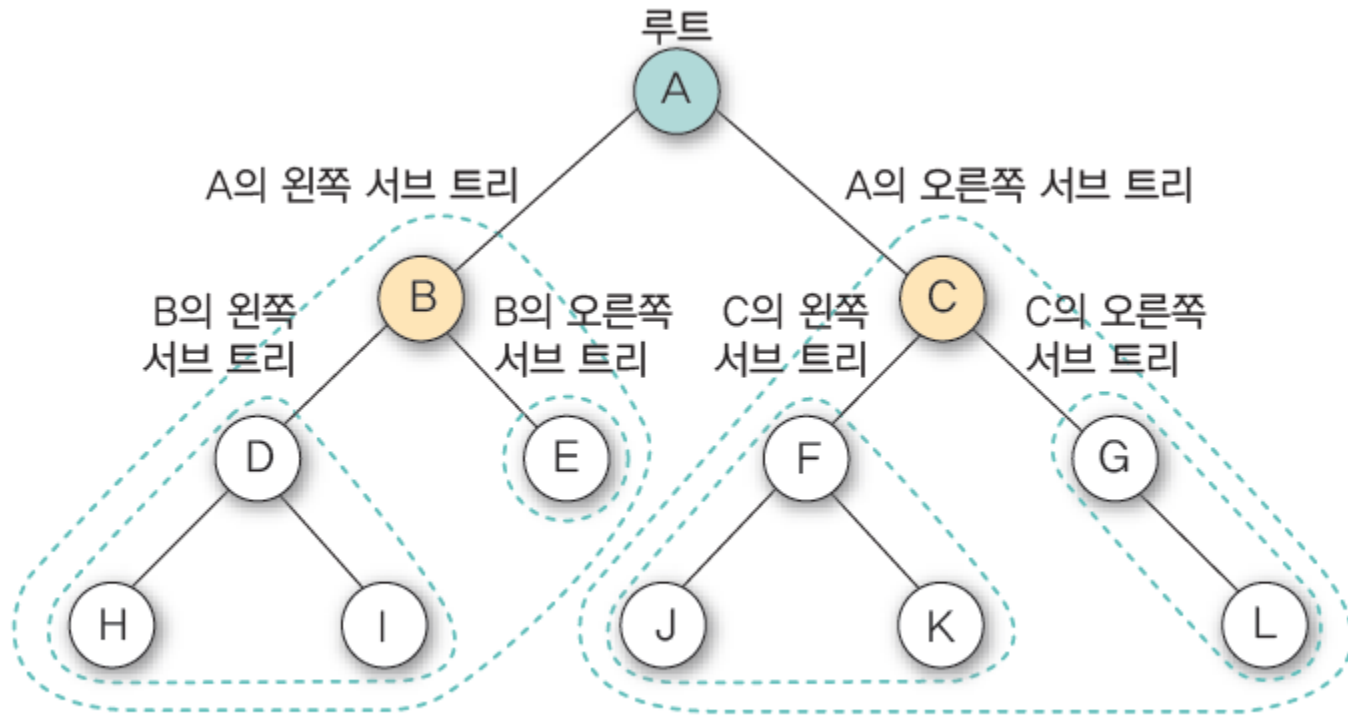
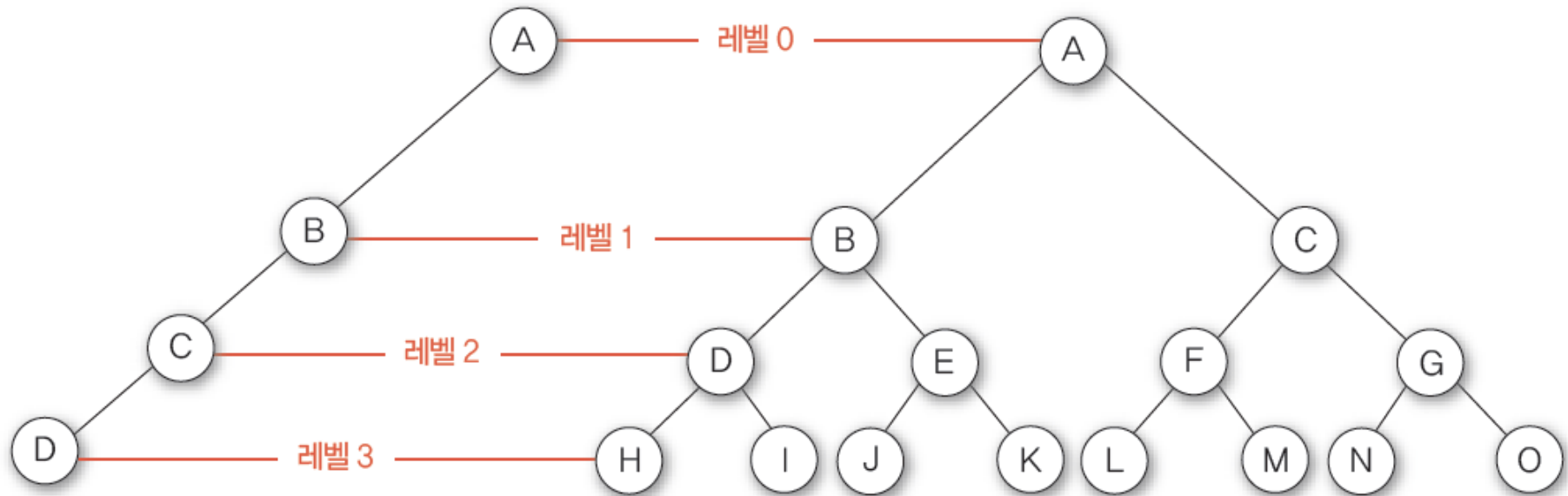


그림 7-5 이진 트리의 서브 트리

2. 이진트리 : 개념과 구조



(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

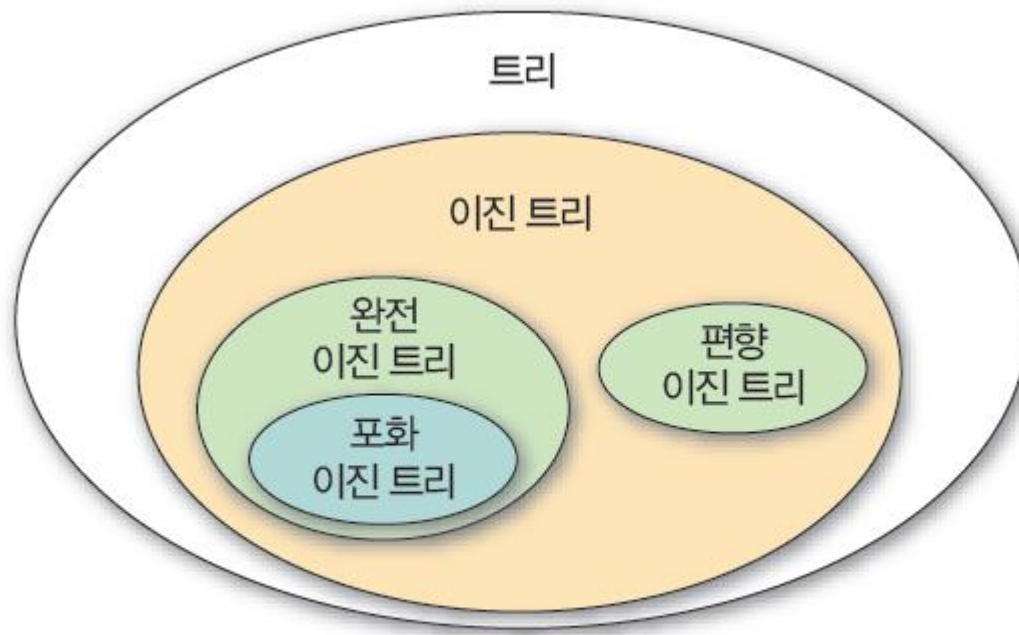


그림 7-8 이진 트리의 종류

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

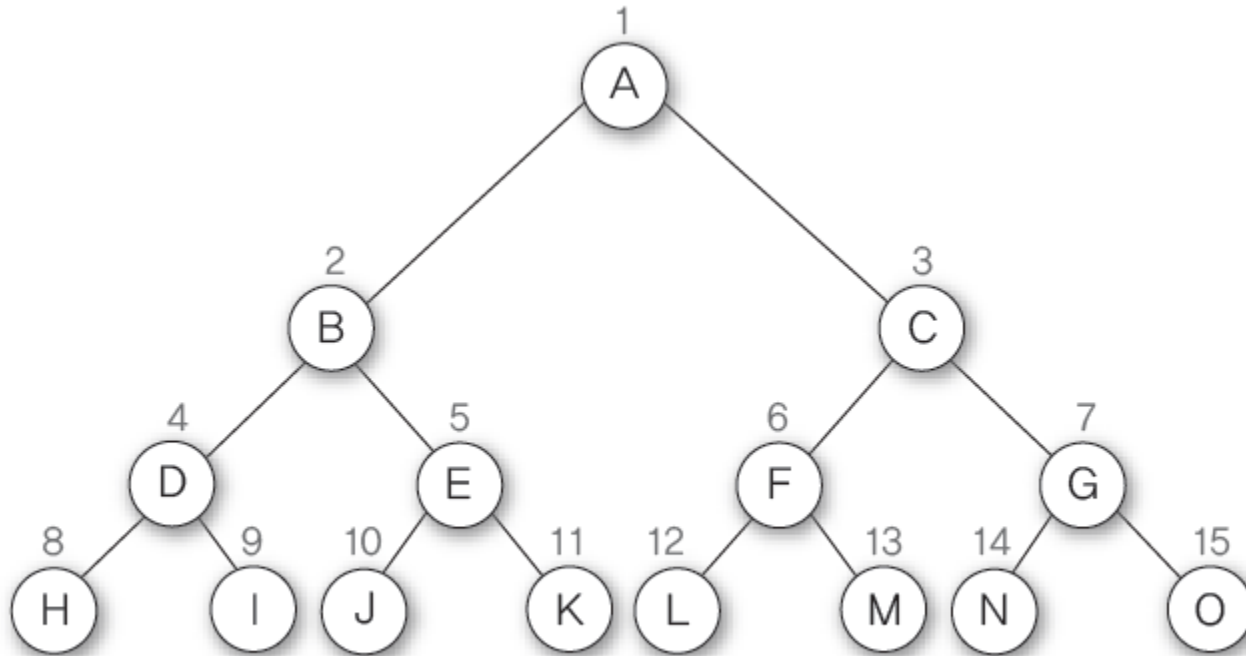


그림 7-9 높이가 3인 포화 이진 트리

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

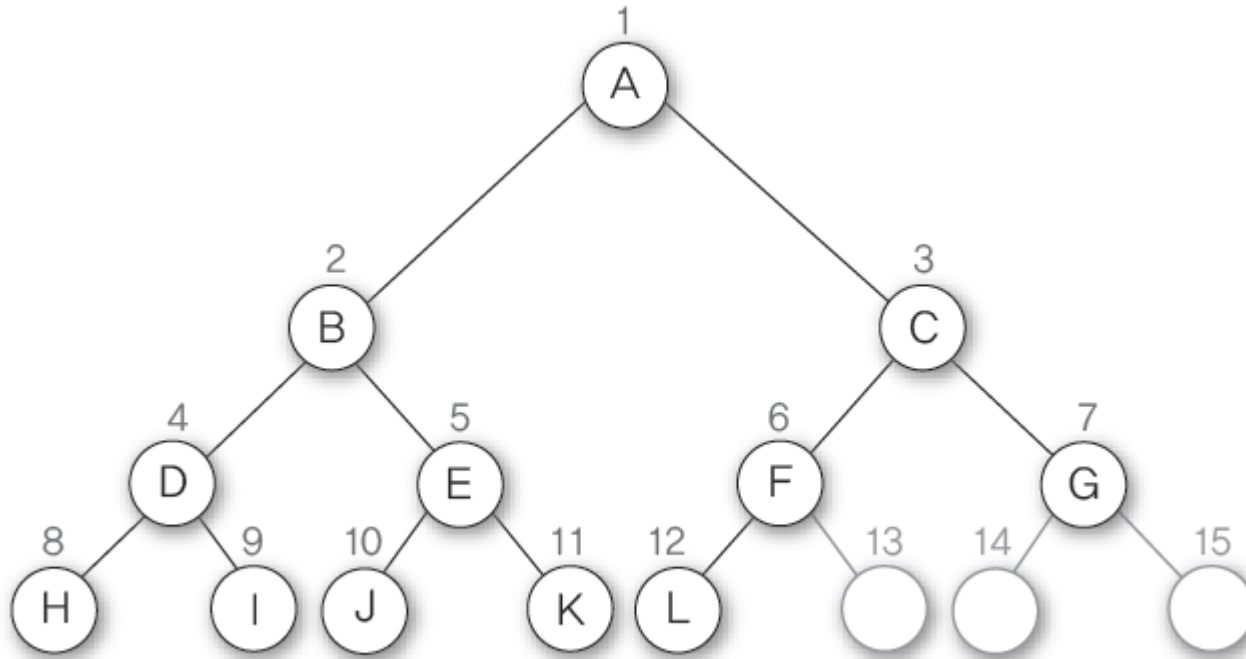
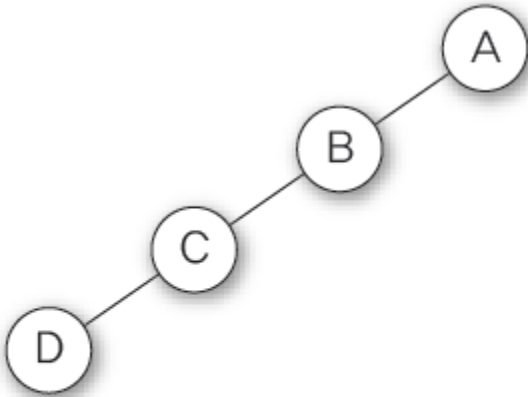


그림 7-10 높이가 3인 완전 이진 트리

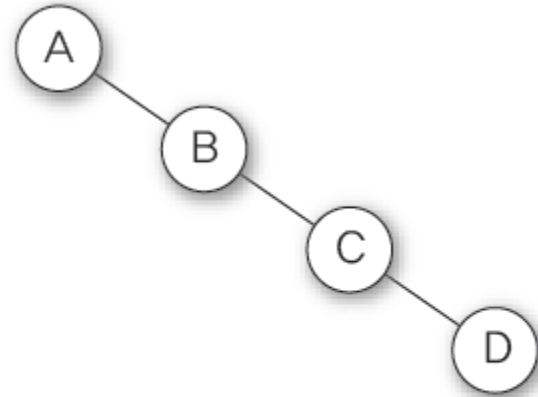
2. 이진트리 : 종류

■ 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 : 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

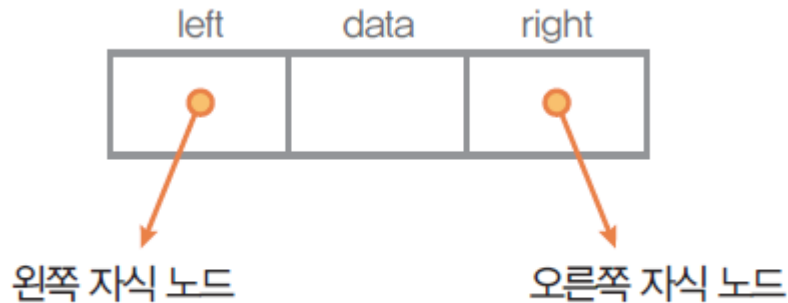


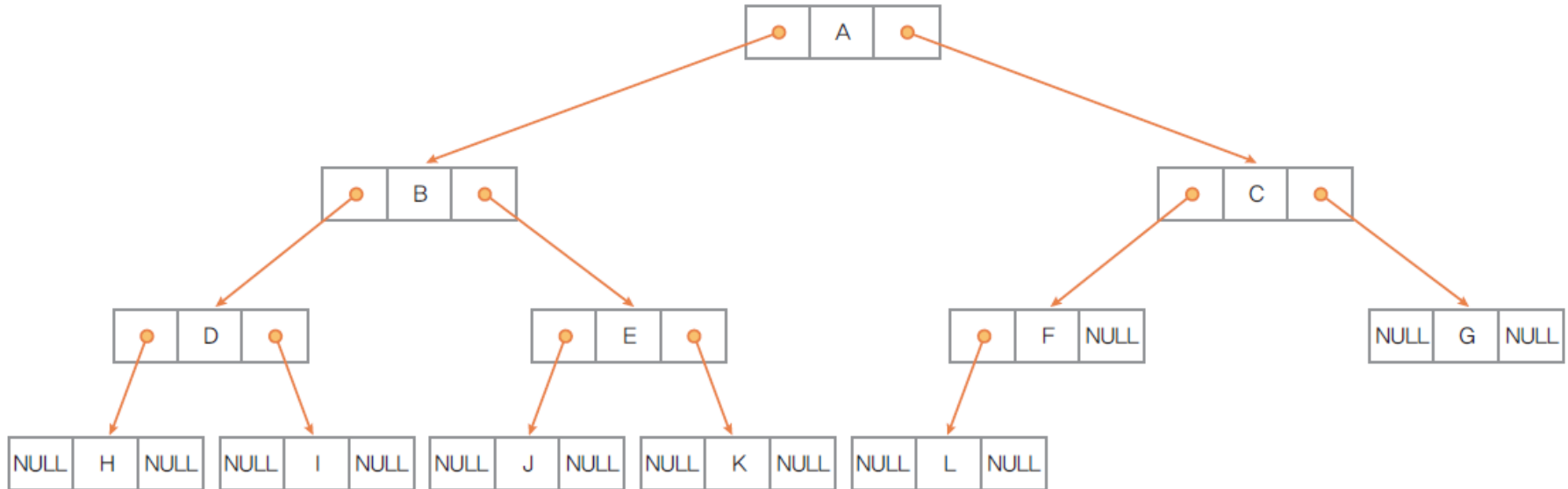
그림 7-14 이진 트리 노드의 구조

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

그림 7-15 이진 트리 노드의 C 구조체 정의

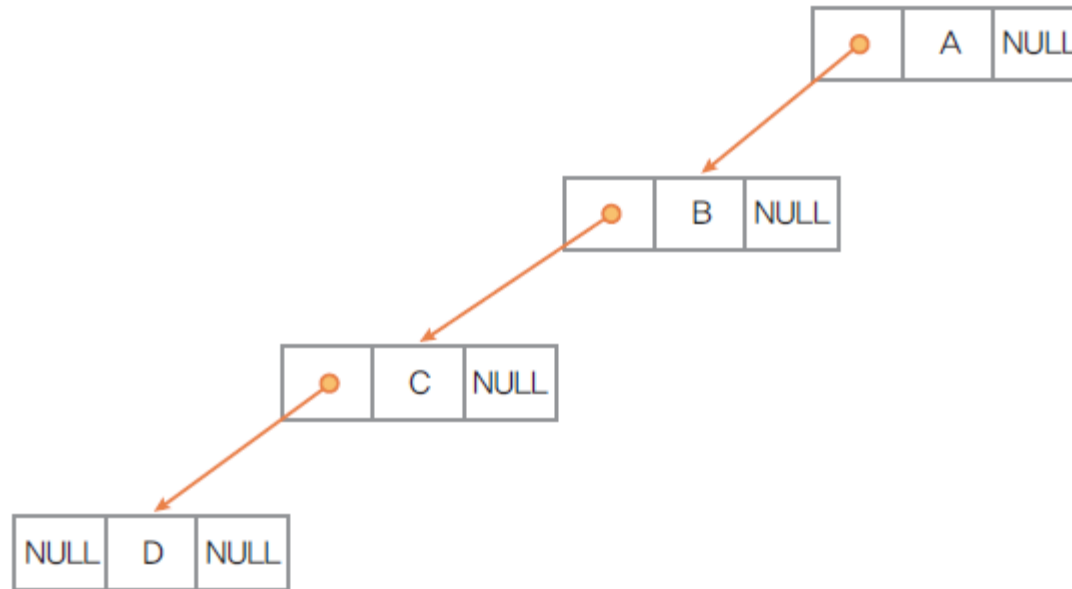
3. 이진트리 구현 : 연결 자료구조를 이용한 구현

- 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 : 개념

● 이진 트리 순회(traversal)의 개념

- 모든 원소를 빠트리거나 중복하지 않고 처리하는 연산을 의미

- 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

- 위의 작업 수행 순서에 따라 전위 순회, 중위 순회, 후위 순회로 나눔.
- 현재 노드에서 서브 트리로 이동하는 작업은 항상 왼쪽 서브 트리 이동(L)을 먼저 하고, 그 다음에 오른쪽 서브 트리 이동(R)함.
- 따라서 전위 순회는 DLR, 중위 순회는 LDR, 후위 순회는 LRD 순서로 순회

4. 이진 트리의 순회 : 개념

● 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 : 개념

■ 전위 순회의 예

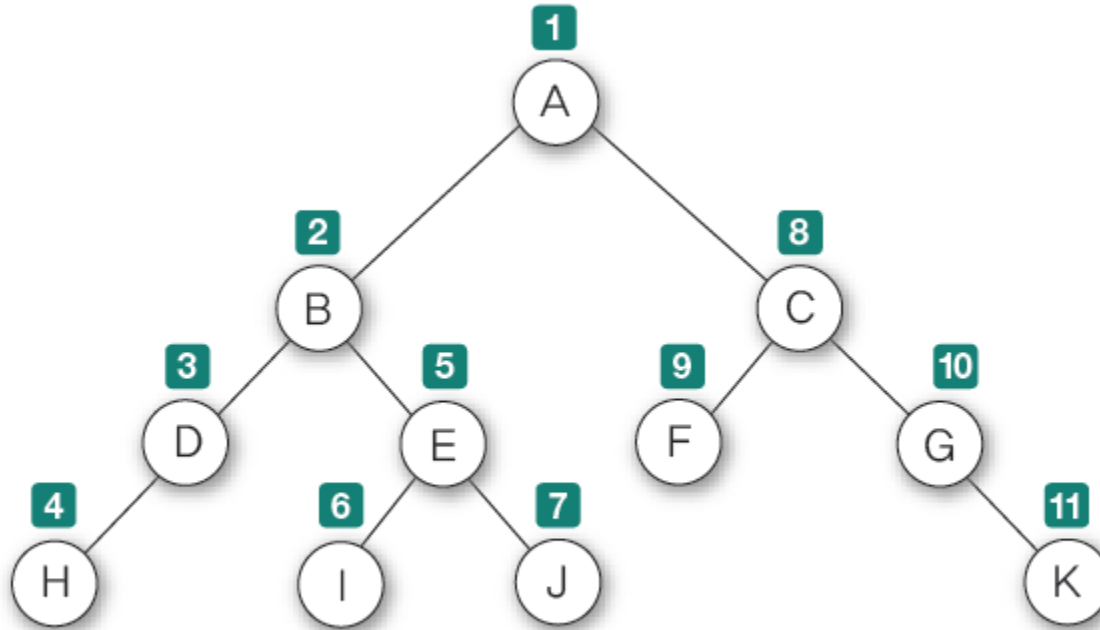


그림 7-19 이진 트리의 전위 순회 경로 : A-B-D-H-E-I-J-C-F-G-K

4. 이진 트리의 순회 : 개념

- ① 노드 A (D-L-R) : 루트 A에서 전위 순회를 시작하여 현재 노드 A의 데이터를 처리. 노드 A (D-L-R) : 왼쪽 서브 트리인 노드 B로 이동
- ② 노드 A (D-L-R) → 노드 B (D-L-R) : 현재 노드 B의 데이터를 처리. 노드 A (D-L-R) → 노드 B (D-L-R) : 왼쪽 서브 트리인 노드 D로 이동
- ③ 노드 A (D-L-R) → 노드 B (D-L-R) → 노드 D (D-L-R) : 현재 노드 D의 데이터를 처리
- ④ 노드 A (D-L-R) → 노드 B (D-L-R) → 노드 D (D-L-R) : 현재 노드 D의 왼쪽 단말 노드 H의 데이터를 처리.
- ⑤ 노드 A (D-L-R) → 노드 B (D-L-R) → 노드 E (D-L-R) : 현재 노드 E의 데이터를 처리
- ⑥ 노드 A (D-L-R) → 노드 B (D-L-R) → 노드 E (D-L-R) : 노드 E의 왼쪽 단말 노드 I의 데이터를 처리
- ⑦ 노드 A (D-L-R) → 노드 B (D-L-R) ← 노드 E (D-L-R) : 노드 E의 오른쪽 단말 노드 J의 데이터를 처리, 부모 노드인 노드 B로 돌아감

4. 이진 트리의 순회 : 개념

- ⑧ 노드 A (D-L-R) : 현재 노드 A의 오른쪽 서브 트리인 노드 C로 이동
- ⑨ 노드 A (D-L-R) → 노드 C (D-L-R) : 현재 노드 C의 왼쪽 단말 노드 F의 데이터를 처리
- ⑩ 노드 A (D-L-R) → 노드 C (D-L-R) → 노드 G (D-L-R) : 현재 노드 G의 데이터를 처리
- ⑪ 노드 A (D-L-R) → 노드 C (D-L-R) → 노드 G (D-L-R) : 노드 G의 왼쪽 노드인 공백 노드를 읽음
노드 A (D-L-R) → 노드 C (D-L-R) ← 노드 G (D-L-R) : 노드 G의 오른쪽 단말 노드 K의 데이터를 처리하고, 이로써 노드 G에서의 D-L-R 순회가 끝났으므로 부모 노드 C로 돌아감
노드 A (D-L-R) ← 노드 C (D-L-R) : 현재 노드 C에서의 D-L-R 순회 역시 끝났으므로, 다시 부모노드 A로 돌아감
노드 A (D-L-R) : 이로써 루트 노드 A에 대한 D-L-R 순회가 끝났으므로 트리 전체에 대한 전위 순회가 완성
전위 순회에 따른 순회 경로는 A-B-D-H-E-I-J-C-F-G-K가 됨

4. 이진트리의 순회 : 개념

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

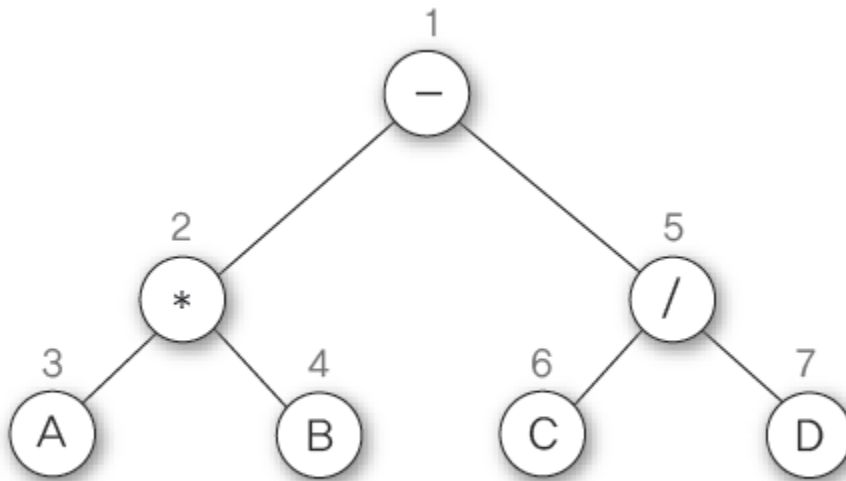


그림 7-20 수식에 대한 이진 트리의 전위 순회 경로: $-*AB/CD$

4. 이진트리의 순회 : 개념

● 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. 이진트리의 순회 : 개념

- 중위 순회의 예

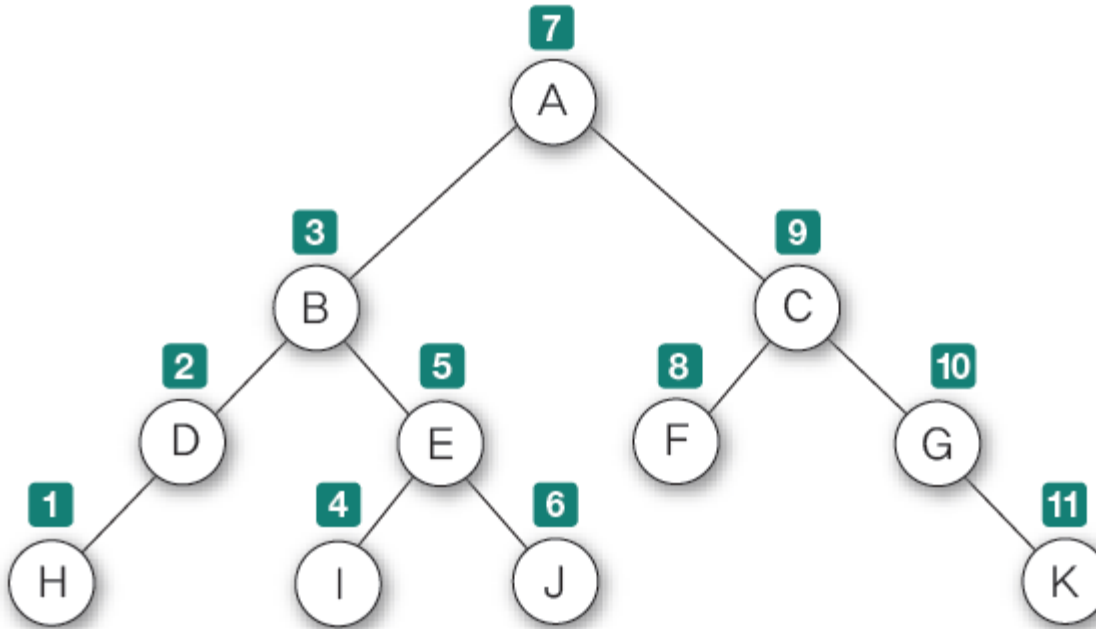


그림 7-22 이진 트리의 중위 순회 경로 : H-D-B-I-E-J-A-F-C-G-K

4. 이진트리의 순회 : 개념

■ 중위 순회에 따른 순회 경로 H-D-B-I-E-J-A-F-C-G-K

- ① A (Ⓐ-D-R) : 루트 A에서 중위 순회를 시작하여, 노드 A의 왼쪽 서브트리 B로 이동
- ② 노드 A (Ⓐ-D-R) → 노드 B (Ⓐ-D-R) → 노드 D (Ⓐ-Ⓓ-R) : 현재 노드 D의 데이터를 처리
- ③ 노드 A (Ⓐ-D-R) → 노드 B (Ⓐ-Ⓓ-R) : 현재 노드 B의 데이터를 처리
- ④ 노드 A (Ⓐ-D-R) → 노드 B (Ⓐ-Ⓓ-Ⓡ) → 노드 E (Ⓐ-D-R) : 현재 노드 E의 왼쪽 단말 노드 I의 데이터를 처리
- ⑤ 노드 A (Ⓐ-D-R) → 노드 B (Ⓐ-Ⓓ-Ⓡ) → 노드 E (Ⓐ-Ⓓ-R) : 현재 노드 E의 데이터를 처리
- ⑥ 노드 A (Ⓐ-D-R) → 노드 B (Ⓐ-Ⓓ-Ⓡ) → 노드 E (Ⓐ-Ⓓ-Ⓡ) : 현재 노드 E의 오른쪽 단말 노드 J의 데이터를 처리
- ⑦ 노드 A (Ⓐ-Ⓓ-R) : 현재 노드 A의 데이터를 처리

4. 이진트리의 순회 : 개념

- ⑧ 노드 A (Ⓛ-ⓓ-Ⓡ) → 노드 C (Ⓛ-ⓓ-Ⓡ) : 현재 노드 C의 왼쪽 단말 노드 F의 데이터를 처리
- ⑨ 노드 A (Ⓛ-ⓓ-Ⓡ) → 노드 C (Ⓛ-ⓓ-Ⓡ) : 현재 노드 C의 데이터 처리
- ⑩ 노드 A (Ⓛ-ⓓ-Ⓡ) → 노드 C (Ⓛ-ⓓ-Ⓡ) → 노드 G (Ⓛ-ⓓ-Ⓡ) : 현재 노드 G의 왼쪽 단말 노드인 공백 노드를 읽음. 노드 A (Ⓛ-ⓓ-Ⓡ) → 노드 C (Ⓛ-ⓓ-Ⓡ) → 노드 G (Ⓛ-ⓓ-Ⓡ) : 노드 G의 데이터를 처리
- ⑪ 노드 A (Ⓛ-ⓓ-Ⓡ) → 노드 C (Ⓛ-ⓓ-Ⓡ) → 노드 G (Ⓛ-ⓓ-Ⓡ) : 현재 노드 G의 오른쪽 단말 노드 K의 데이터를 처리. 노드 A (Ⓛ-ⓓ-Ⓡ) → 노드 C (Ⓛ-ⓓ-Ⓡ) ← 노드 G (Ⓛ-ⓓ-Ⓡ) : 노드 G의 L-D-R 순회가 끝났으므로 부모 노드 C로 돌아감. 노드 A (Ⓛ-ⓓ-Ⓡ) ← 노드 C (Ⓛ-ⓓ-Ⓡ) : 현재 노드 C의 L-D-R 순회가 끝났으므로 부모 노드 A로 돌아감. 노드 A (Ⓛ-ⓓ-Ⓡ) : 이로써 루트 노드 A에 대한 L-D-R 순회를 모두 끝냈으므로 트리 전체에 대한 중위 순회가 완성

4. 이진트리의 순회 : 개념

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

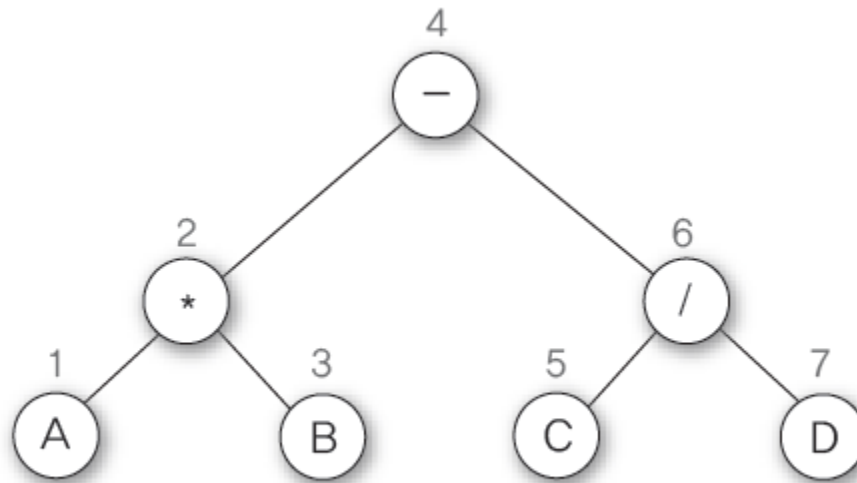


그림 7-23 수식에 대한 이진 트리의 중위 순회 경로 : $A*B-C/D$

4. 이진트리의 순회 : 개념

● 후위 순회(postorder traversal)

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

- ① 작업 L : 현재 노드 n 의 왼쪽 서브 트리로 이동한다.
- ② 작업 R : 현재 노드 n 의 오른쪽 서브 트리로 이동한다.
- ③ 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. 이진트리의 순회 : 개념

- 후위 순회의 예

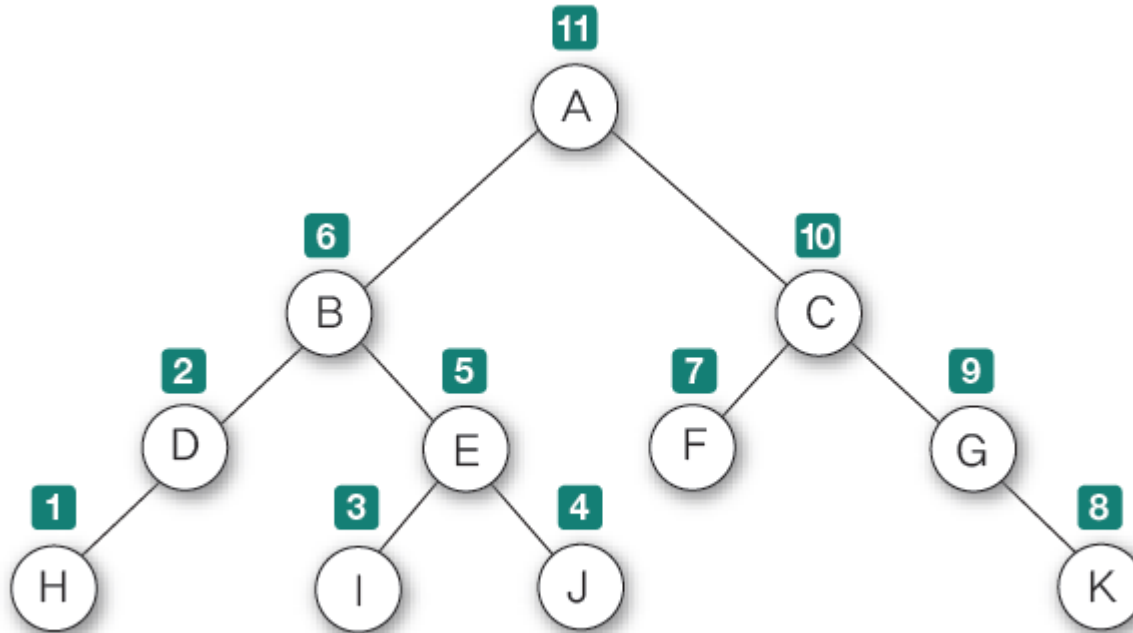


그림 7-25 이진 트리의 후위 순회 경로 : H-D-I-J-E-B-F-K-G-C-A

4. 이진트리의 순회 : 개념

- 후위 순회에 따른 순회 경로 H-D-I-J-E-B-F-K-G-C-A

- ① 노드 A (\textcircled{L} -R-D) : 루트 A에서 후위 순회를 시작하여 노드 A의 왼쪽 서브 트리 B로 이동
- ② 노드 A (\textcircled{L} -R-D) → 노드 B (\textcircled{L} -R-D) → 노드 D (\textcircled{L} - \textcircled{R} -D) : 노드 D의 오른쪽 단말 노드인 공백 노드를 읽음
- ③ 노드 A (\textcircled{L} -R-D) → 노드 B (\textcircled{L} - \textcircled{R} -D) : 현재 노드 B의 오른쪽 서브 트리 E로 이동
- ④ 노드 A (\textcircled{L} -R-D) → 노드 B (\textcircled{L} - \textcircled{R} -D) → 노드 E (\textcircled{L} - \textcircled{R} -D) : 노드 E의 오른쪽 단말 노드 J의 데이터를 처리
- ⑤ 노드 A (\textcircled{L} -R-D) → 노드 B (\textcircled{L} - \textcircled{R} -D) ← 노드 E (\textcircled{L} - \textcircled{R} - \textcircled{D}) : 현재 노드 E의 데이터를 처리하고, 부모 노드 B로 돌아감
- ⑥ 노드 A (\textcircled{L} -R-D) ← 노드 B (\textcircled{L} - \textcircled{R} - \textcircled{D}) : 현재 노드 B의 데이터를 처리하고, 부모 노드 A로 돌아감
- ⑦ 노드 A (\textcircled{L} - \textcircled{R} -D) → 노드 C (\textcircled{L} -R-D) : 현재 노드 C의 왼쪽 단말 노드 F의 데이터를 처리

4. 이진트리의 순회 : 개념

- ⑧ 노드 A (\textcircled{L} - \textcircled{R} -D) \rightarrow 노드 C (\textcircled{L} - \textcircled{R} -D) : 현재 노드 C의 오른쪽 서브 트리 G로 이동
노드 A (\textcircled{L} - \textcircled{R} -D) \rightarrow 노드 C (\textcircled{L} - \textcircled{R} -D) \rightarrow 노드 G (\textcircled{L} -R-D) : 현재 노드 G의 왼쪽 단말 노드인 공백 노드를 읽음
노드 A (\textcircled{L} - \textcircled{R} -D) \rightarrow 노드 C (\textcircled{L} - \textcircled{R} -D) \rightarrow 노드 G (\textcircled{L} - \textcircled{R} -D) : 노드 G의 오른쪽 단말 노드 K의 데이터를 처리
- ⑨ 노드 A (\textcircled{L} - \textcircled{R} -D) \rightarrow 노드 C (\textcircled{L} - \textcircled{R} -D) \leftarrow 노드 G (\textcircled{L} - \textcircled{R} - \textcircled{D}) : 현재 노드 G의 데이터를 처리하고, 부모 노드 C로 돌아감
- ⑩ 노드 A (\textcircled{L} - \textcircled{R} - D) \leftarrow 노드 C (\textcircled{L} - \textcircled{R} - \textcircled{D}) : 현재 노드 C의 데이터를 처리하고, 부모 노드 A로 이동
- ⑪ 노드 A (\textcircled{L} - \textcircled{R} - \textcircled{D}) : 현재 노드 A의 데이터를 처리한다. 이로써 루트 노드 A에 대한 L-R-D 순회가 끝났으므로 트리 전체에 대한 후위 순회가 완성

4. 이진트리의 순회 : 개념

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

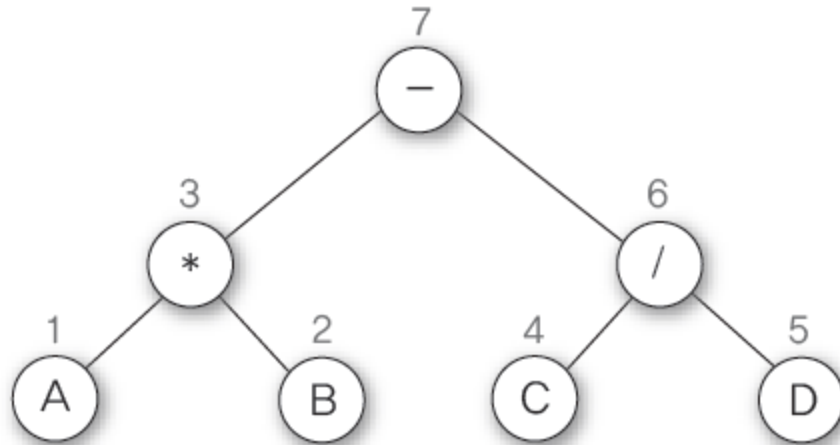


그림 7-26 수식에 대한 이진 트리의 후위 순회 경로 : $AB*CD/-$

4. 이진 트리의 순회 : 순회의 구현

● 수식 이진 트리 순회 프로그램

- 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

예제 7-1

이진 트리 순회하기

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include <memory.h>
04
05  typedef struct treeNode {    // 연결 자료구조로 구성하기 위해 트리의 노드 정의
06      char data;
07      struct treeNode *left;   // 왼쪽 서브 트리에 대한 링크 필드
08      struct treeNode *right;  // 오른쪽 서브 트리에 대한 링크 필드
09  } treeNode;
10
```

4. 이진 트리의 순회 : 순회의 구현

```
11 // data를 루트 노드로 하여 왼쪽 서브 트리와 오른쪽 서브 트리를 연결하는 연산
12 treeNode* makeRootNode(char data, treeNode* leftNode, treeNode* rightNode) {
13     treeNode* root = (treeNode *)malloc(sizeof(treeNode));
14     root->data = data;
15     root->left = leftNode;
16     root->right = rightNode;
17     return root;
18 }
19
20 // 이진 트리에 대한 전위 순회 연산 [알고리즘 7-1] 구현
21 void preorder(treeNode* root) {
22     if (root) {
23         printf("%c", root->data);
24         preorder(root->left);
25         preorder(root->right);
26     }
27 }
28
```

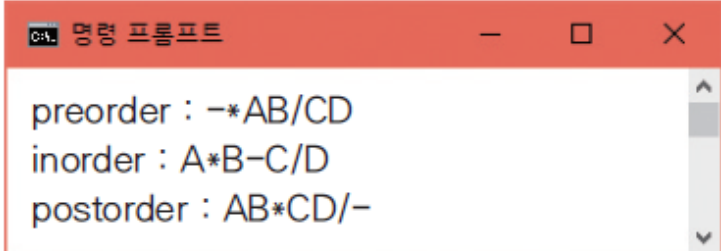
4. 이진 트리의 순회 : 순회의 구현

```
29 // 이진 트리에 대한 중위 순회 연산 [알고리즘 7-2] 구현
30 void inorder(treeNode* root) {
31     if (root) {
32         inorder(root->left);
33         printf("%c", root->data);
34         inorder(root->right);
35     }
36 }
```

```
37
38 // 이진 트리에 대한 후위 순회 연산 [알고리즘 7-3] 구현
39 void postorder(treeNode* root) {
40     if (root) {
41         postorder(root->left);
42         postorder(root->right);
43         printf("%c", root->data);
44     }
45 }
```

4. 이진 트리의 순회 : 순회의 구현

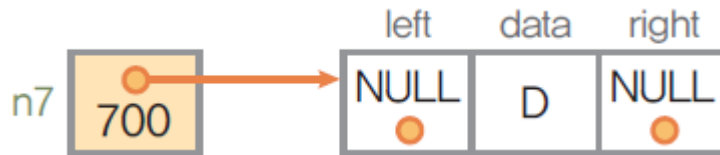
```
47 void main() {
48     // 수식 (A*B-C/D)를 이진 트리로 만들기
49     treeNode* n7 = makeRootNode('D', NULL, NULL);
50     treeNode* n6 = makeRootNode('C', NULL, NULL);
51     treeNode* n5 = makeRootNode('B', NULL, NULL);
52     treeNode* n4 = makeRootNode('A', NULL, NULL);
53     treeNode* n3 = makeRootNode('/', n6, n7);
54     treeNode* n2 = makeRootNode('*', n4, n5);
55     treeNode* n1 = makeRootNode('-', n2, n3);
56
57     printf("\n preorder : ");
58     preorder(n1);
59
60     printf("\n inorder : ");
61     inorder(n1);
62     printf("\n postorder : ");
63     postorder(n1);
64
65     getchar();
66 }
```



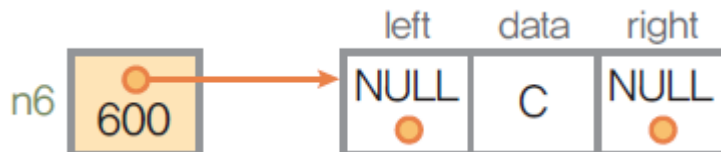
```
C:\> 명령 프롬프트
preorder : -*AB/CD
inorder : A*B-C/D
postorder : AB*CD/-
```

4. 이진 트리의 순회 : 순회의 구현

- 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 트리와 오른쪽 서브 트리가 공백 노드인 이진 트리 노드 n7을 선언

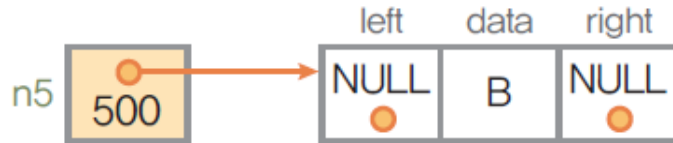


- 50행 : 데이터 필드가 'C'이고 왼쪽 서브 트리와 오른쪽 서브 트리가 공백 노드인 이진 트리 노드 n6을 만듦

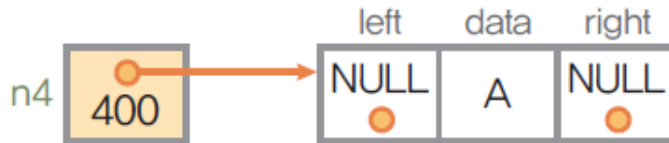


4. 이진 트리의 순회 : 순회의 구현

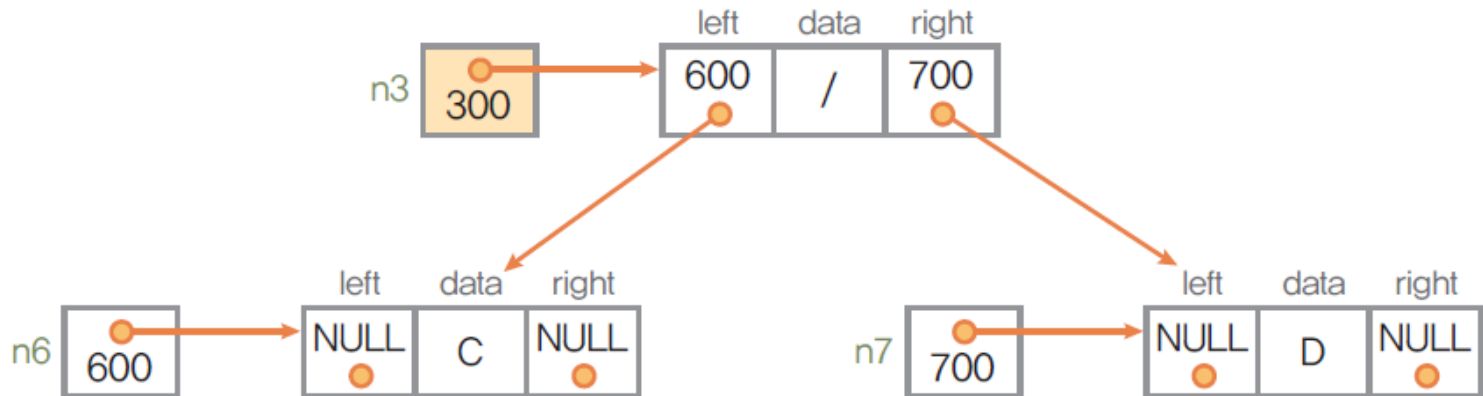
- 51행 : 데이터 필드가 B' '이고 왼쪽 서브 트리과 오른쪽 서브 트리가 공백 노드인 이진 트리 노드 n5 를 만듦



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

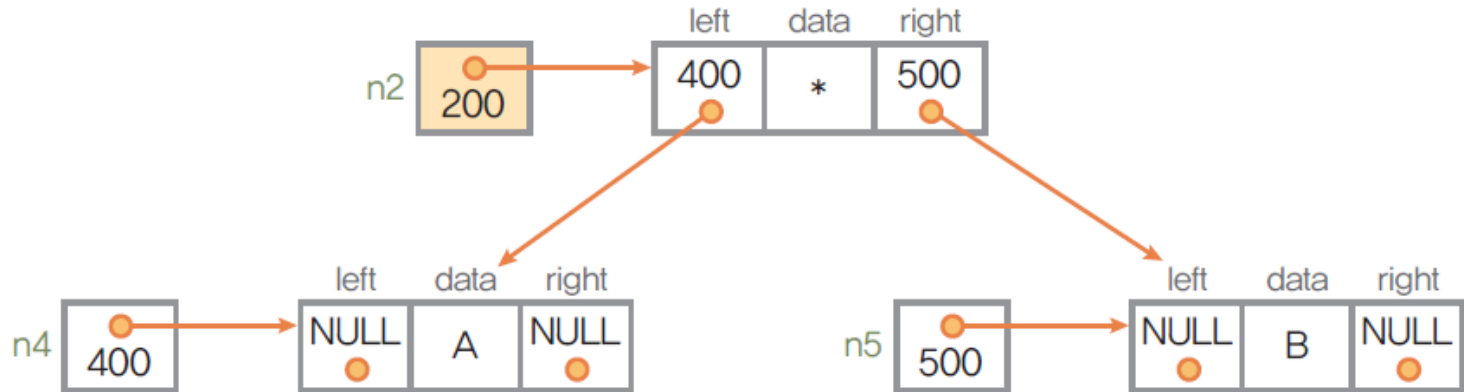


- 53행 : 데이터 필드가 '/'이고, 왼쪽 링크 필드에 노드 n6, 오른쪽 링크 필드에 노드 n7이 연결된 이진 트리 노드 n3을 만듦

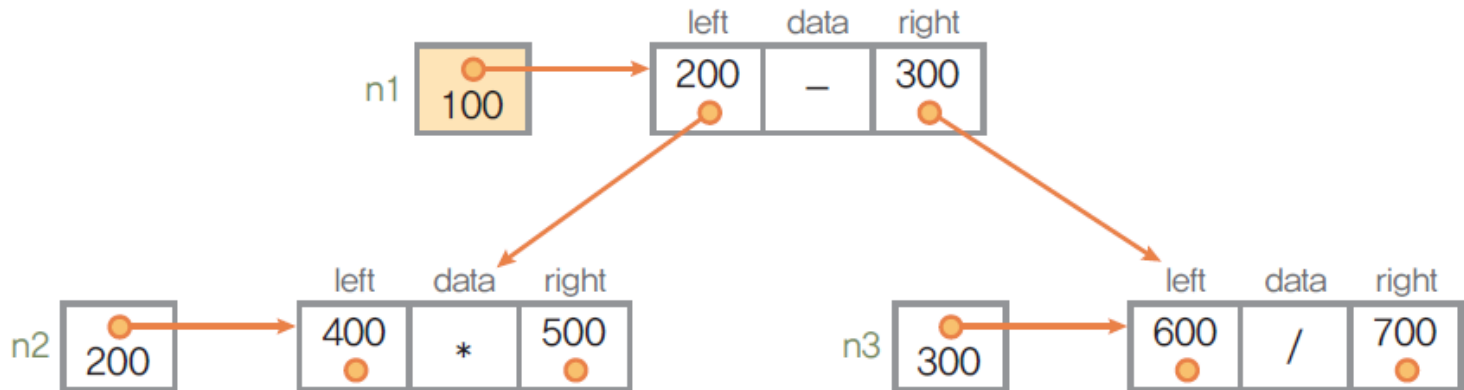


4. 이진 트리의 순회 : 순회의 구현

- 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 : 순회의 구현

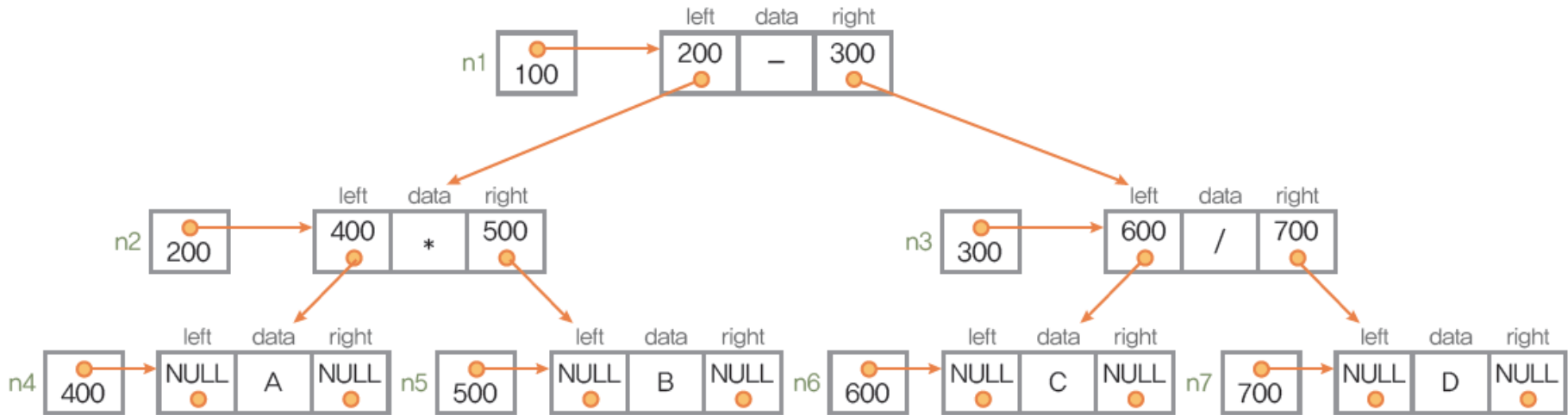


그림 7-27 `makeRootNode()`를 이용한 이진 트리의 구성 과정

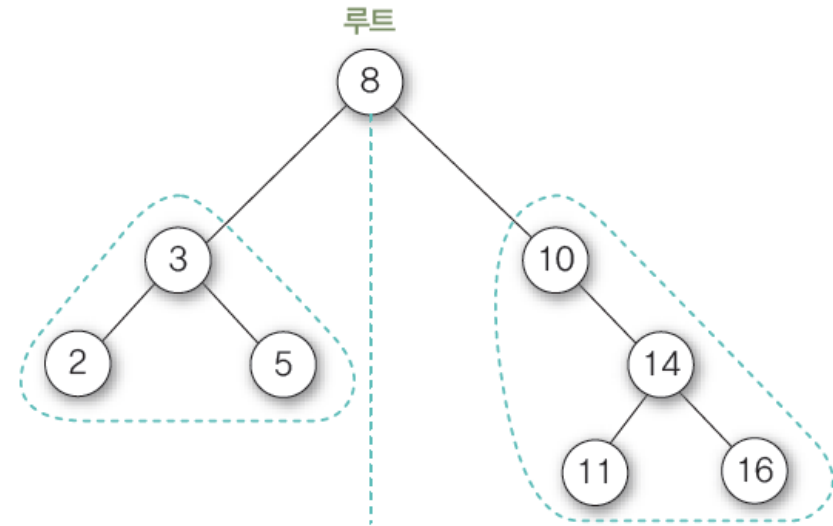
5. 이진 탐색 트리 : 개념

● 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

5. 이진 탐색 트리 : 탐색 연산

● 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 : 탐색 연산

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 : 탐색 연산

- 이진 탐색 트리에서 원소 11을 탐색

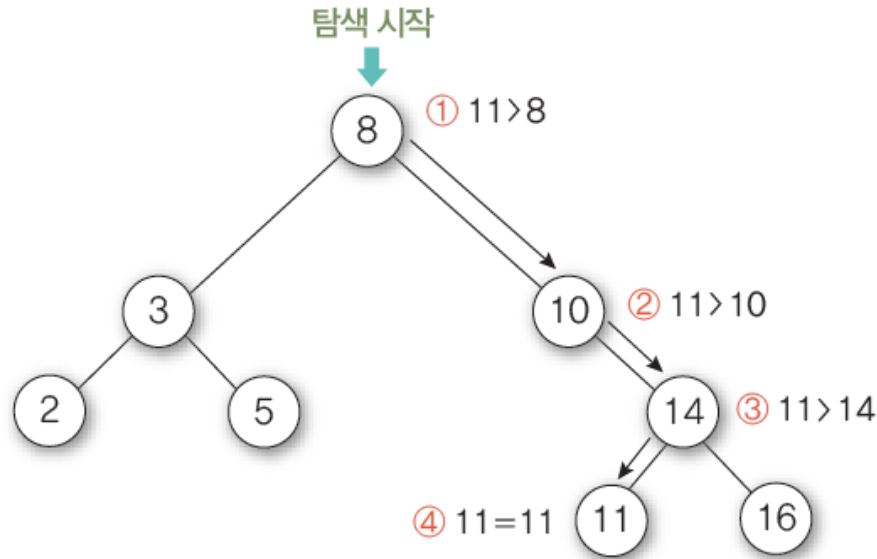


그림 7-34 이진 탐색 트리에서의 탐색 과정 예

- 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

5. 이진 탐색 트리 : 삽입 연산

● 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 : 삽입 연산

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

```
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 : 삽입 연산

- 이진 탐색 트리에 원소 4를 삽입 하기

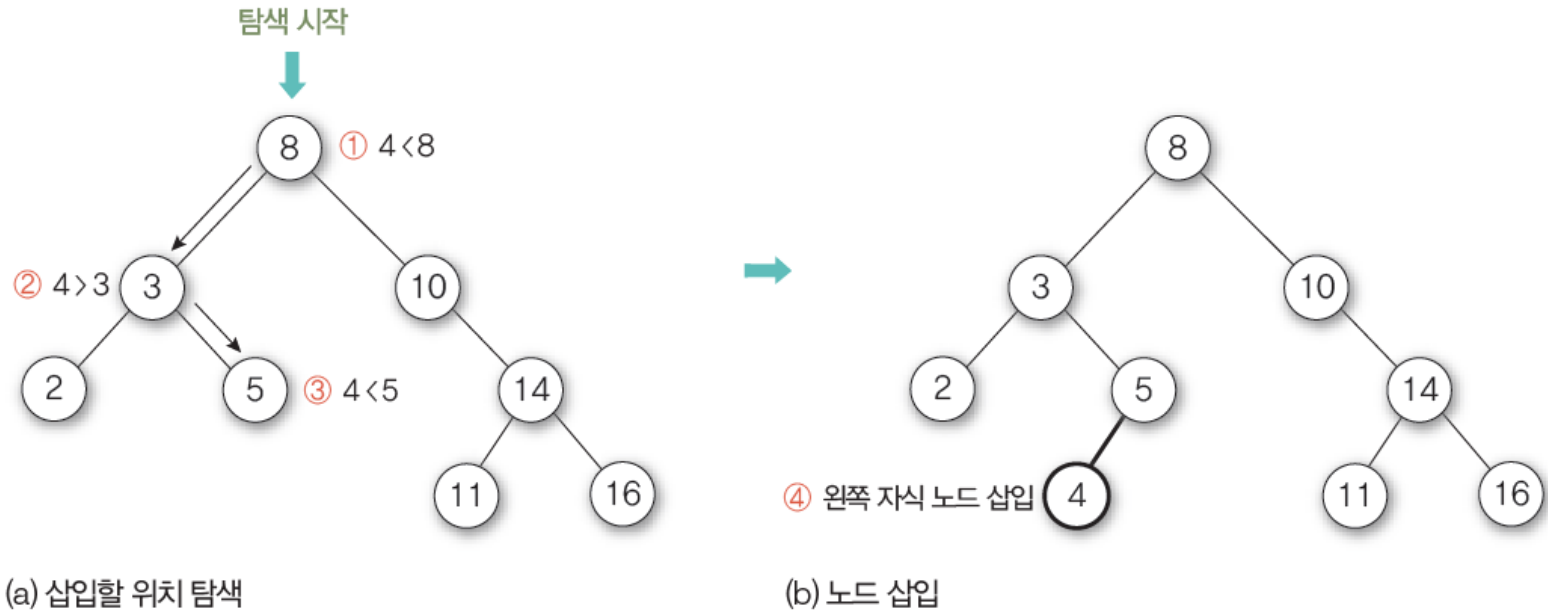


그림 7-35 이진 탐색 트리에서의 삽입 과정 예

- 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 : 삽입 연산

- 이진 탐색 트리에 원소 4를 삽입 하는 과정을 연결 자료구조로 표현

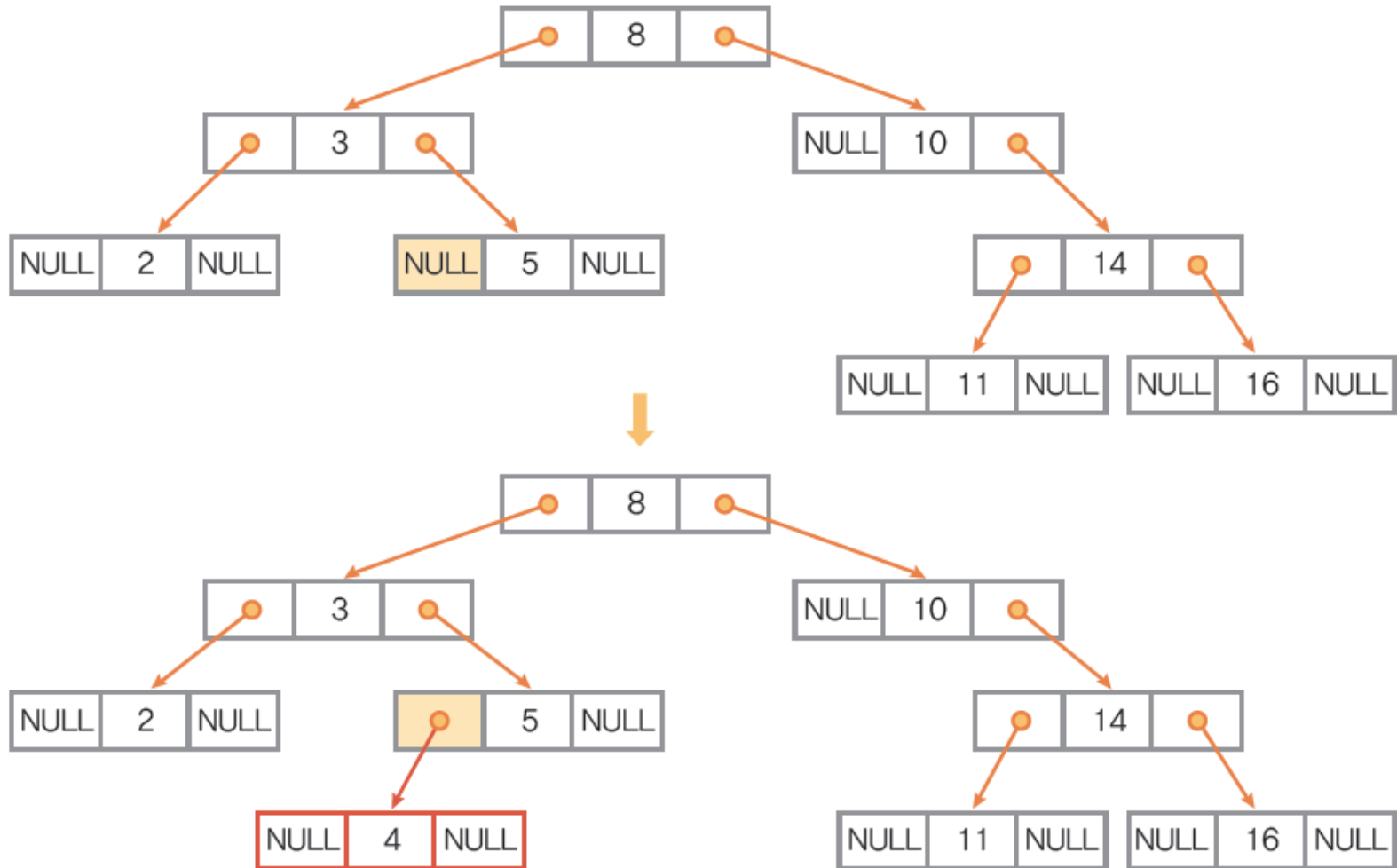


그림 7-36 이진 탐색 트리에 원소 4를 삽입하기 전과 후로 나눠 연결 자료구조로 표현

5. 이진 탐색 트리 : 삭제 연산

● 이진 탐색 트리의 삭제 연산

1) 먼저 탐색 연산을 수행

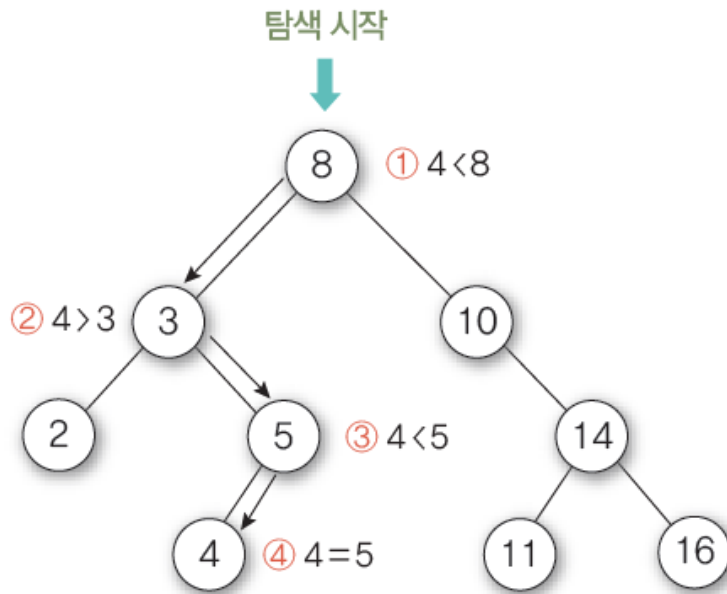
- 삭제할 노드의 위치를 알아야 하므로 트리를 탐색

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)

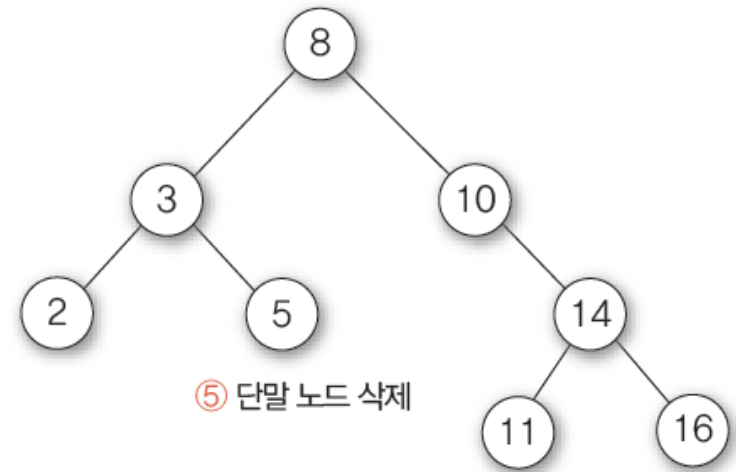
5. 이진 탐색 트리 : 삭제 연산

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



(a) 삭제할 노드 탐색

그림 7-37 이진 탐색 트리에서 단말 노드 4를 삭제하는 예



(b) 노드 4 삭제

5. 이진 탐색 트리 : 삭제 연산

- 4를 삭제하기 전과 후로 연결 자료구조를 표현

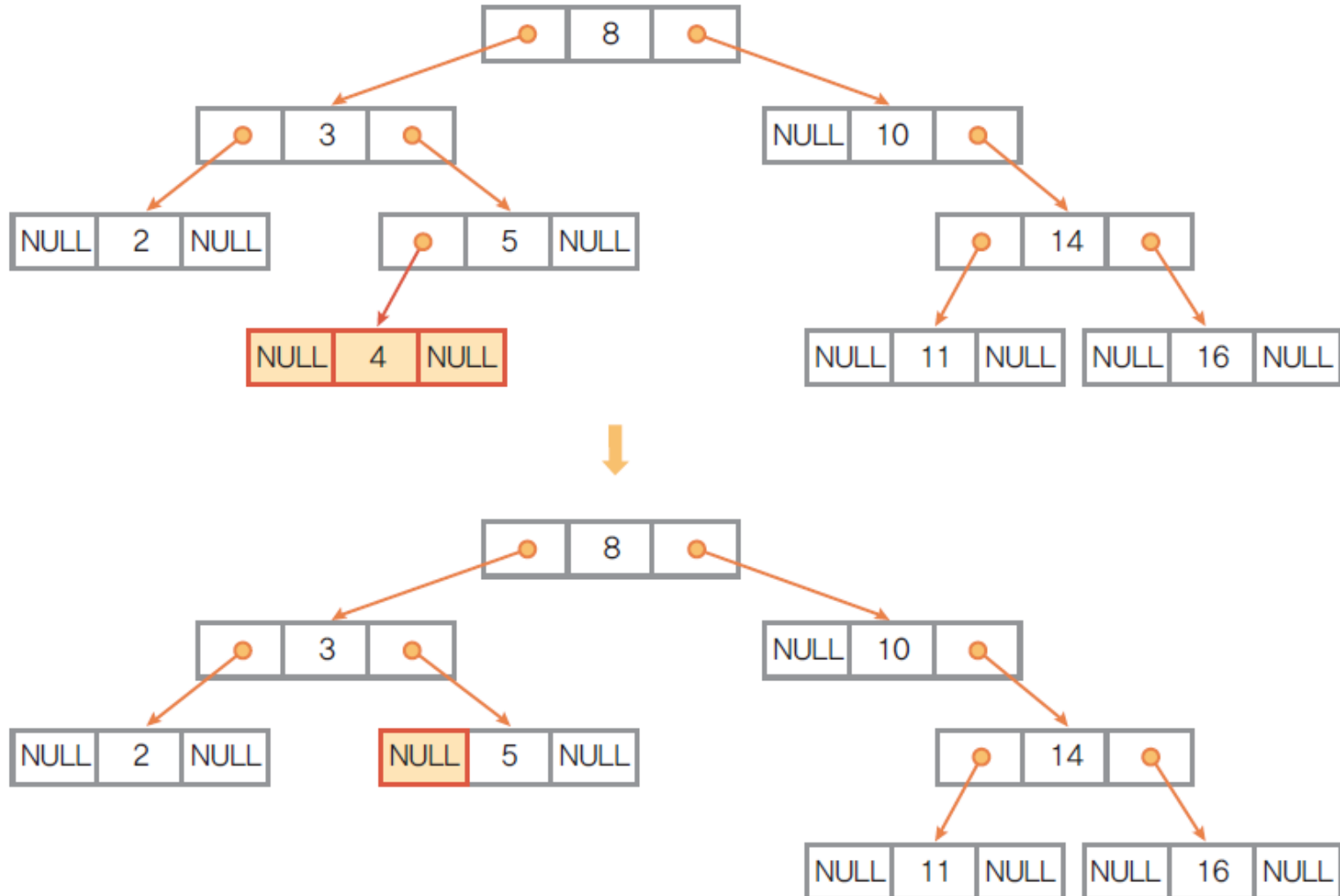
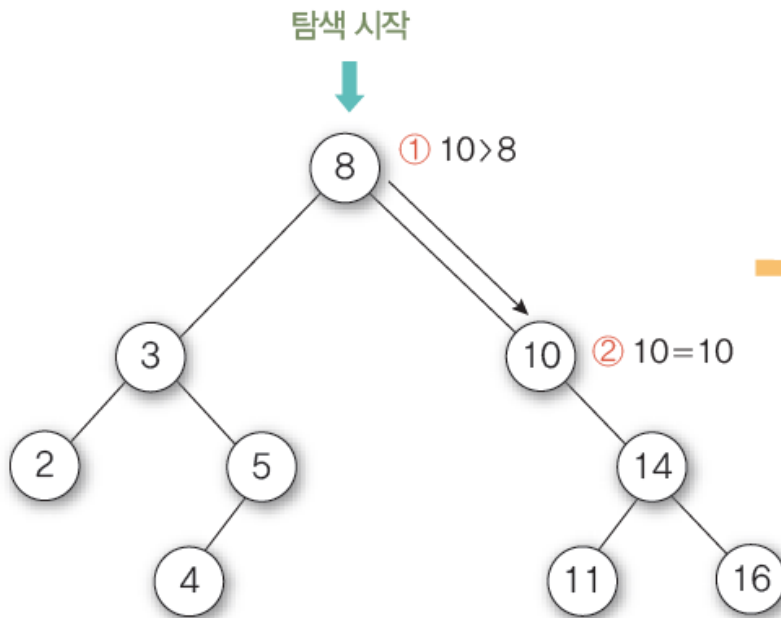


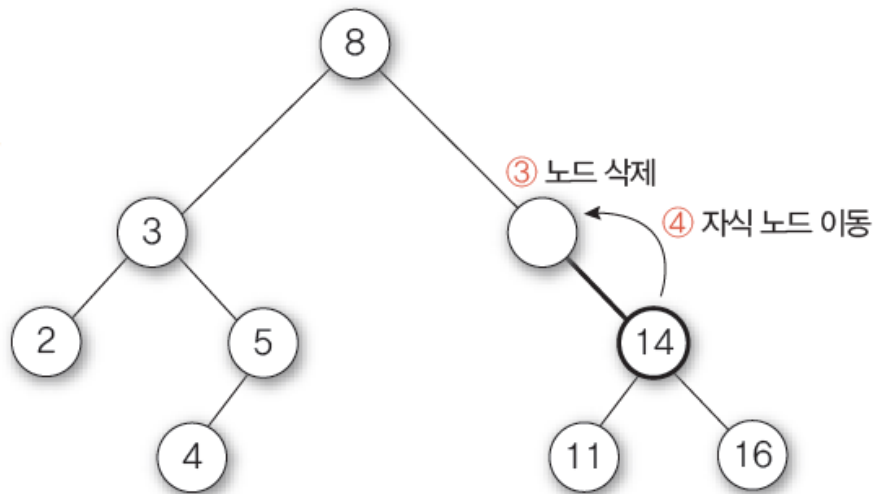
그림 7-38 이진 탐색 트리에서 단말 노드 4를 삭제하기 전과 후로 나눠 연결 자료구조로 표현

5. 이진 탐색 트리 : 삭제 연산

- 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

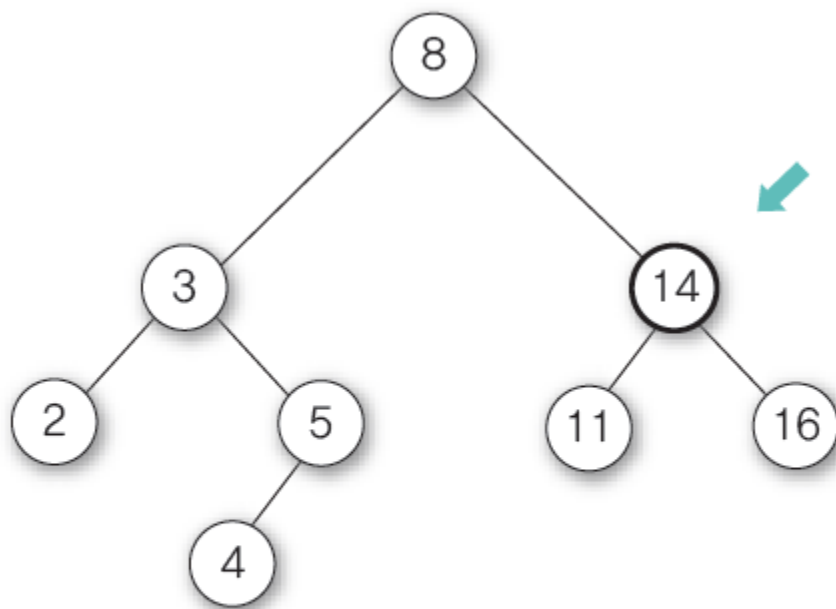


(a) 삭제할 노드 탐색



(b) 노드 삭제

5. 이진 탐색 트리 : 삭제 연산



(c) 자식 노드의 위치 조정

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

5. 이진 탐색 트리 : 삭제 연산

- 노드 10을 삭제하는 경우에 대한 단순 연결 리스트 표현

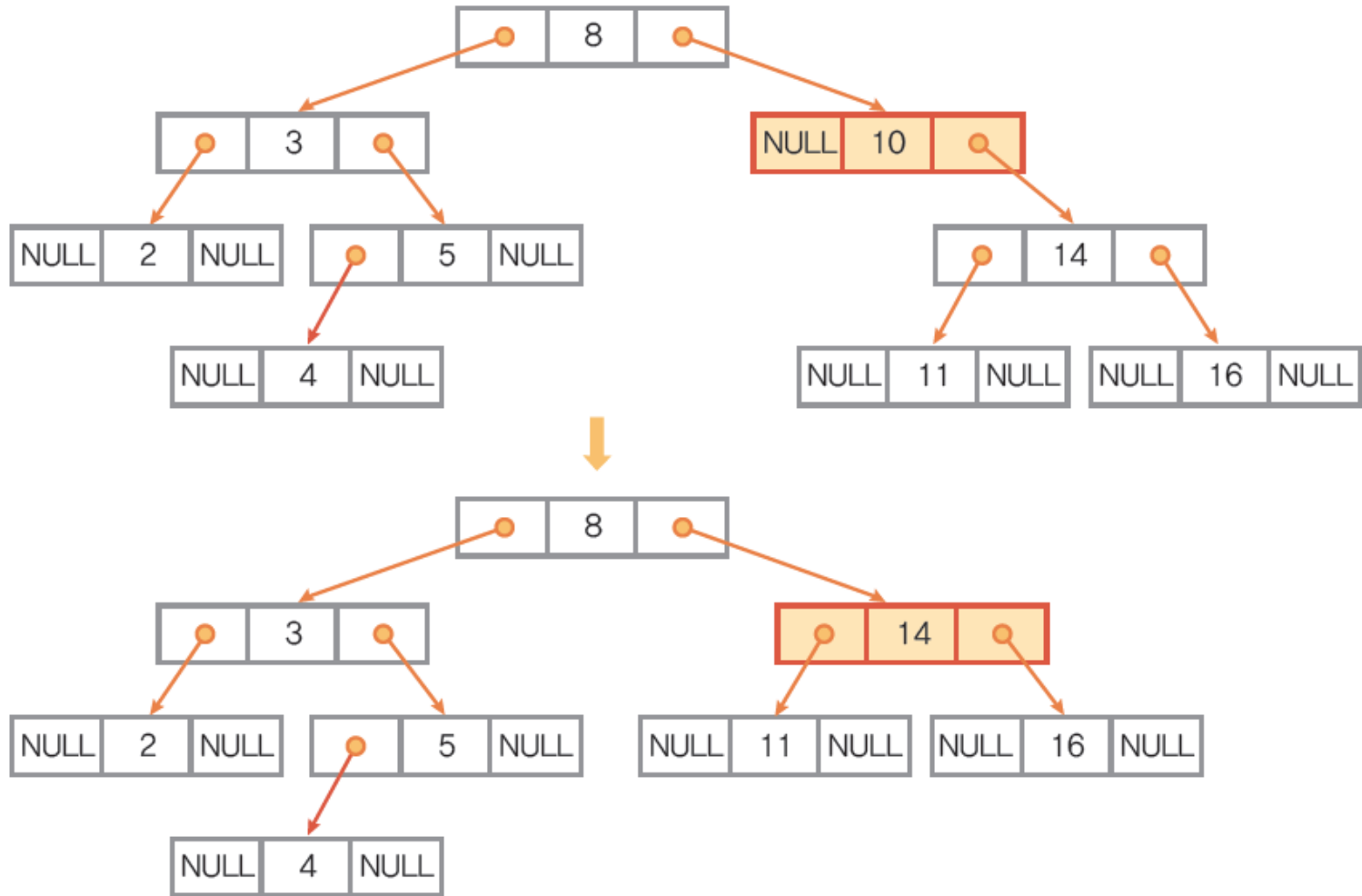


그림 7-40 이진 탐색 트리에서 자식 노드가 하나인 노드 10을 삭제하기 전과 후로 나눠 연결 자료구조로 표현

5. 이진 탐색 트리 : 삭제 연산

- 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 : 삭제 연산

- 삭제한 노드의 자리를 물려받을 수 있는 자손 노드

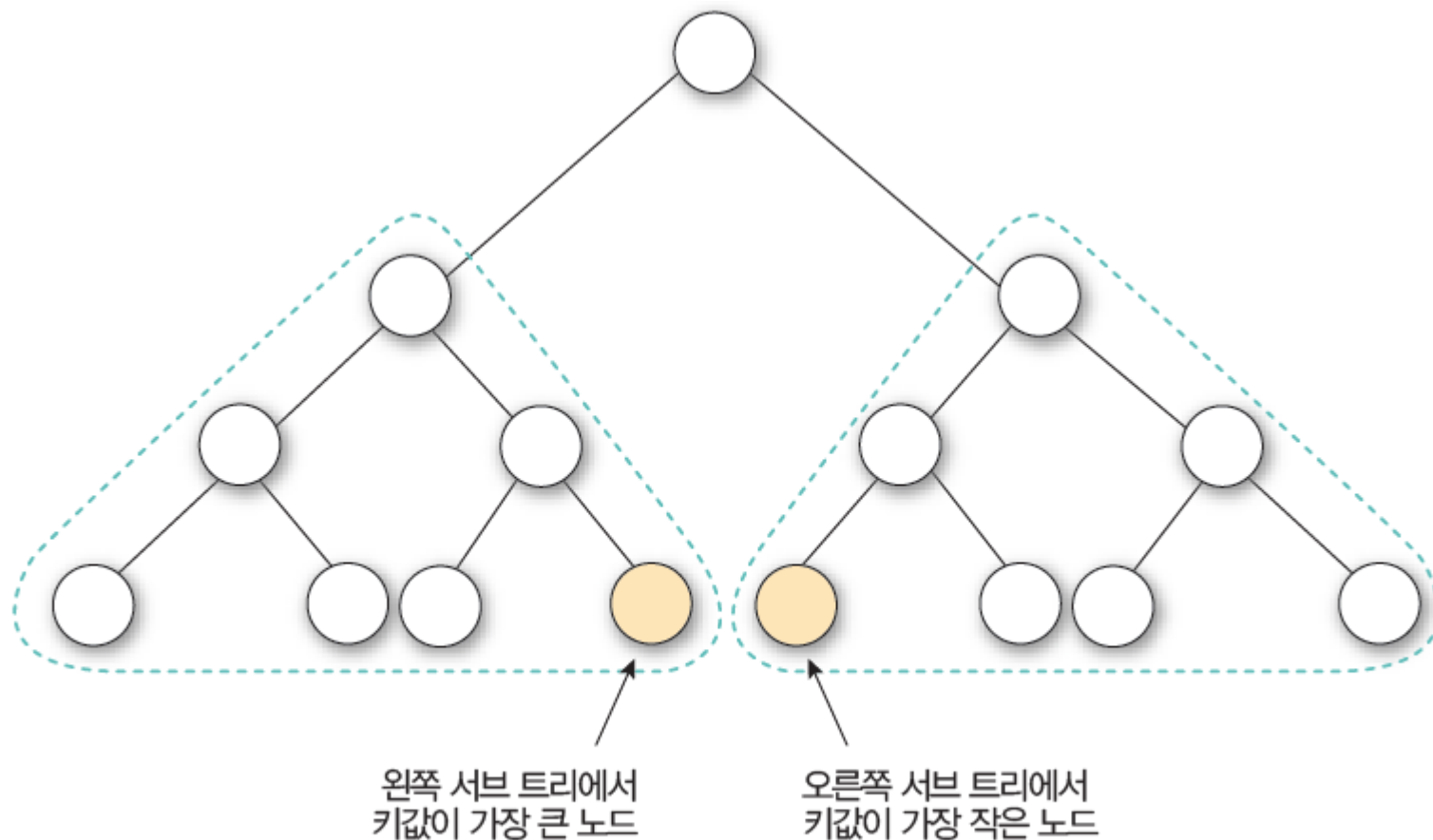
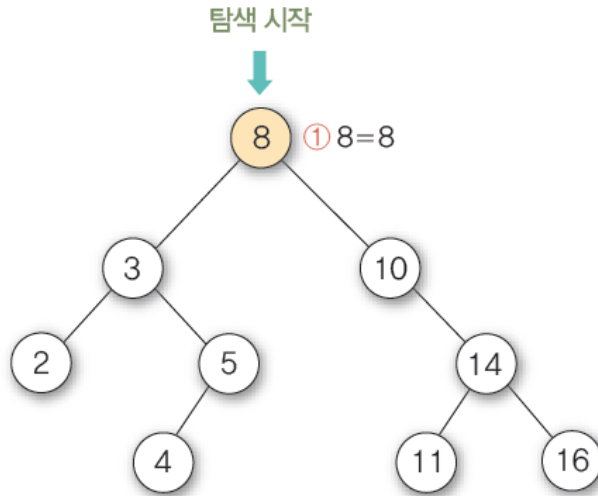


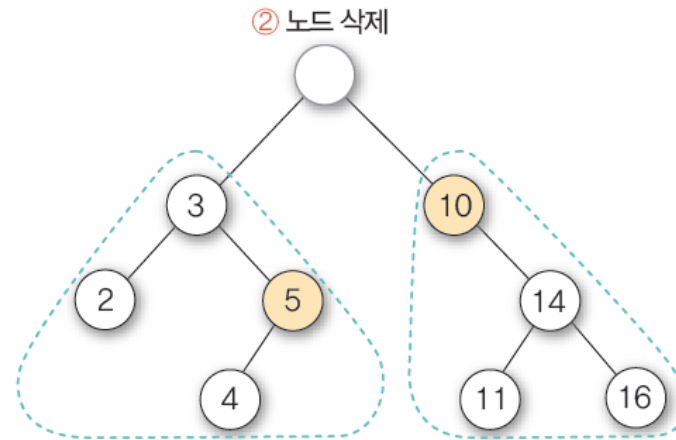
그림 7-41 삭제할 노드의 자리를 물려받을 수 있는 자손 노드

5. 이진 탐색 트리 : 삭제 연산

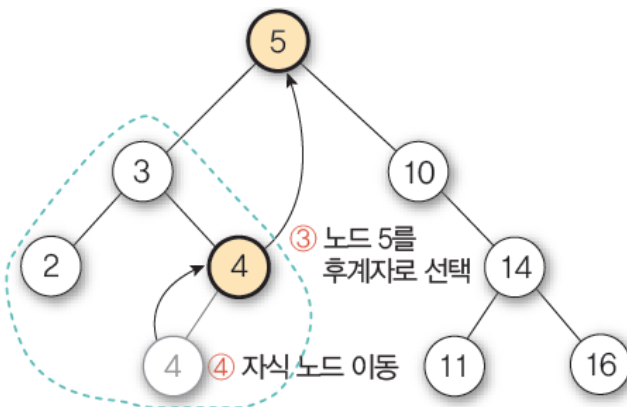
- 노드 8을 삭제하는 경우



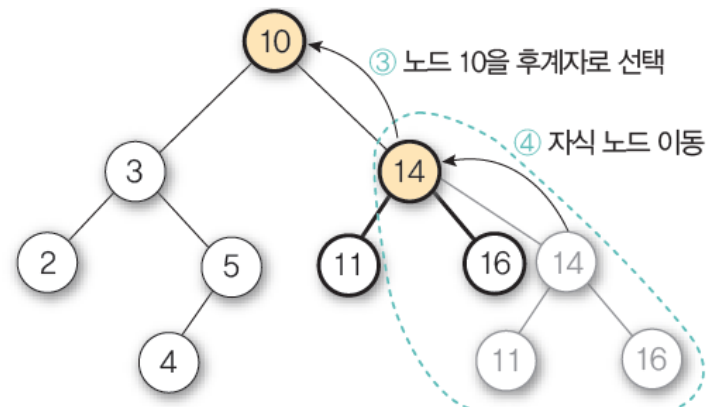
(a) 삭제할 노드 탐색



(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 : 삭제 연산

- 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 : 삭제 연산

■ 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 : 삭제 연산

```
// 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 : 구현

● 이진 탐색 트리의 구현

- 이진 탐색 트리를 연결 자료구조를 사용하여 표현하고 이진 탐색 트리에 대한 연산을 수행하는 C 프로그램

예제 7-4

이진 탐색 트리의 연산 프로그램

```
001  #include<stdio.h>
002  #include<stdlib.h>
003
004  typedef char element;      // 이진 탐색 트리 element의 자료형을 char로 정의
005  typedef struct treeNode {
006      char key;              // 데이터 필드
007      struct treeNode* left; // 왼쪽 서브 트리 링크 필드
008      struct treeNode* right; // 오른쪽 서브 트리 링크 필드
009  } treeNode;
010
011  // 이진 탐색 트리에서 키값이 x인 노드의 위치를 탐색하는 연산 [알고리즘 7-4] 구현
012  treeNode* searchBST(treeNode* root, char x) {
013      treeNode* p;
014      p = root;
```

5. 이진 탐색 트리 : 구현

```
015 while (p != NULL) {
016     if (x < p->key) p = p->left;
017     else if (x == p->key) return p;
018     else p = p->right;
019 }
020 printf("\n 찾는 키가 없습니다!");
021 return p;
022 }
```

```
023
024 // 포인터 p가 가리키는 노드와 비교하여 노드 x를 삽입하는 연산 [알고리즘 7-5] 수정 구현
025 treeNode* insertNode(treeNode *p, char x) {
026     treeNode *newNode;
027     if (p == NULL) {
028         newNode = (treeNode*)malloc(sizeof(treeNode));
029         newNode->key = x;
030         newNode->left = NULL;
031         newNode->right = NULL;
032         return newNode;
033     }
```


5. 이진 탐색 트리 : 구현

```
034     else if (x < p->key) p->left = insertNode(p->left, x);
035     else if (x > p->key) p->right = insertNode(p->right, x);
036     else printf("\n 이미 같은 키가 있습니다! \n");
037
038     return p;
039 }
040
041 // 루트 노드부터 탐색하여 키값과 같은 노드를 찾아 삭제하는 연산 [알고리즘 7-6] 수정 구현
042 void deleteNode(treeNode *root, element key) {
043     treeNode *parent, *p, *succ, *succ_parent;
044     treeNode *child;
045
046     parent = NULL;
047     p = root;
048     while ((p != NULL) && (p->key != key)) { // 삭제할 노드의 위치 탐색
049         parent = p;
050         if (key < p->key) p = p->left;
051         else p = p->right;
052     }
053 }
```

5. 이진 탐색 트리 : 구현

```
054 // 삭제할 노드가 없는 경우
055 if (p == NULL) {
056     printf("\n 찾는 키가 이진 트리에 없습니다!!");
057     return;
058 }
059
060 // 삭제할 노드가 단말 노드인 경우
061 if ((p->left == NULL) && (p->right == NULL)) {
062     if (parent != NULL) {
063         if (parent->left == p) parent->left = NULL;
064         else parent->right = NULL;
065     }
066     else root = NULL;
067 }
068
069 // 삭제할 노드가 자식 노드를 한 개 가진 경우
070 else if ((p->left == NULL) || (p->right == NULL)) {
071     if (p->left != NULL) child = p->left;
072     else child = p->right;
```

5. 이진 탐색 트리 : 구현

```
073
074     if (parent != NULL) {
075         if (parent->left == p) parent->left = child;
076         else parent->right = child;
077     }
078     else root = child;
079 }
080
081 // 삭제할 노드가 자식 노드를 두 개 가진 경우
082 else {
083     succ_parent = p;
084     succ = p->left;
085     while (succ->right != NULL) {           // 왼쪽 서브 트리에서 후계자 찾기
086         succ_parent = succ;
087         succ = succ->right;
088     }
089     if (succ_parent->left == succ) succ_parent->left = succ->left;
090     else succ_parent->right = succ->left;
091     p->key = succ->key;
```

5. 이진 탐색 트리 : 구현

```
092     p = succ;
093 }
094 free(p);
095 }
096
097 // 이진 탐색 트리를 중위 순회하면서 출력하는 연산
098 void displayInorder(treeNode* root) {
099     if (root) {
100         displayInorder(root->left);
101         printf("%c_", root->key);
102         displayInorder(root->right);
103     }
104 }
105
106 void menu() {
107     printf("\n*-----*");
108     printf("\n\t1 : 트리 출력");
109     printf("\n\t2 : 문자 삽입");
```

5. 이진 탐색 트리 : 구현

```
110     printf("\n\t3 : 문자 삭제");
111     printf("\n\t4 : 문자 검색");
112     printf("\n\t5 : 종료");
113     printf("\n*-----*");
114     printf("\n메뉴 입력 >> ");
115 }
116
117 int main() {
118     treeNode* root = NULL;
119     treeNode* foundedNode = NULL;
120     char choice, key;
121
122     // [그림 7-43]과 같은 초기 이진 탐색 트리를 구성하고
123     // 노드 G를 루트 노드 포인터 root로 지정
124     root = insertNode(root, 'G');
125     insertNode(root, 'I');
126     insertNode(root, 'H');
127     insertNode(root, 'D');
```

5. 이진 탐색 트리 : 구현

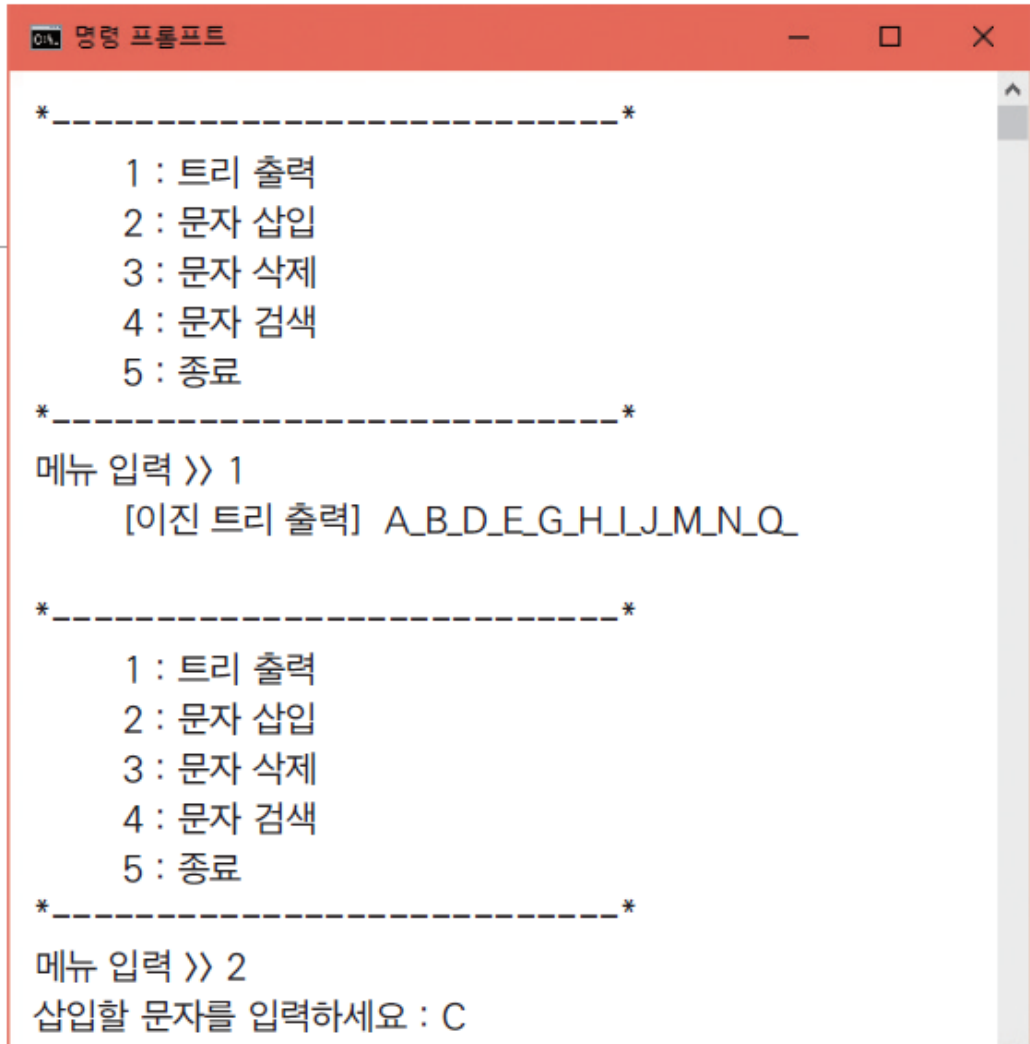
```
128     insertNode(root, 'B');
129     insertNode(root, 'M');
130     insertNode(root, 'N');
131     insertNode(root, 'A');
132     insertNode(root, 'J');
133     insertNode(root, 'E');
134     insertNode(root, 'Q');
135
136     while (1) {
137         menu();
138         scanf(" %c", &choice);
139
140         switch (choice - '0') {
141             case 1: printf("\t[이진 트리 출력] ");
142                     displayInorder(root); printf("\n");
143                     break;
144
145             case 2: printf("삽입할 문자를 입력하세요 : ");
146                     scanf(" %c", &key);
147                     insertNode(root, key);
148                     break;
```

5. 이진 탐색 트리 : 구현

```
149
150     case 3: printf("삭제할 문자를 입력하세요 : ");
151             scanf(" %c", &key);
152             deleteNode(root, key);
153             break;
154
155     case 4: printf("찾을 문자를 입력하세요 : ");
156             scanf(" %c", &key);
157             foundedNode = searchBST(root, key);
158             if (foundedNode != NULL)
159                 printf("\n %c를 찾았습니다! \n", foundedNode->key);
160             else printf("\n 문자를 찾지 못했습니다. \n");
161             break;
162
163     case 5: return 0;
164
```

5. 이진 탐색 트리 : 구현

```
165         default: printf("없는 메뉴입니다. 메뉴를 다시 선택하세요! \n");
166             break;
167     }
168 }
169 }
```



The screenshot shows a Windows command prompt window titled "명령 프롬프트" (Command Prompt). The program displays a menu with five options: 1: 트리 출력 (Tree Output), 2: 문자 삽입 (Character Insertion), 3: 문자 삭제 (Character Deletion), 4: 문자 검색 (Character Search), and 5: 종료 (Exit). The user enters '1', and the program outputs the binary search tree structure: A_B_D_E_G_H_I_J_M_N_Q_. The user then enters '2', and the program prompts for a character to insert, with 'C' entered.

```
C:\> 명령 프롬프트

*-----*
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_Q_

*-----*
1 : 트리 출력
2 : 문자 삽입
3 : 문자 삭제
4 : 문자 검색
5 : 종료
*-----*
메뉴 입력 >> 2
삽입할 문자를 입력하세요 : C
```


5. 이진 탐색 트리 : 구현

```
C:\ 명령 프롬프트

*-----*
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 >>
```

5. 이진 탐색 트리 : 구현

- 011~022행의 searchBST()는 이진 탐색 트리에서 키값이 x인 노드의 위치를 탐색하는 연산을 수행
- 024~039행의 insertNode()는 포인터 p 노드와 비교하여 단말 노드 x를 삽입하는 연산을 수행
- 041~095행의 deleteNode()는 이진 탐색 트리의 루트 노드부터 탐색하여 키값과 같은 노드를 찾아 삭제하는 연산을 수행
- 097~104행은 이진 탐색 트리를 중위 순회하면서 출력하는 연산을 수행
- 122~134행은 이진 탐색 트리를 구성하고 첫 번째로 삽입한 노드, 즉 노드 G를 루트 노드 포인터 root로 지정

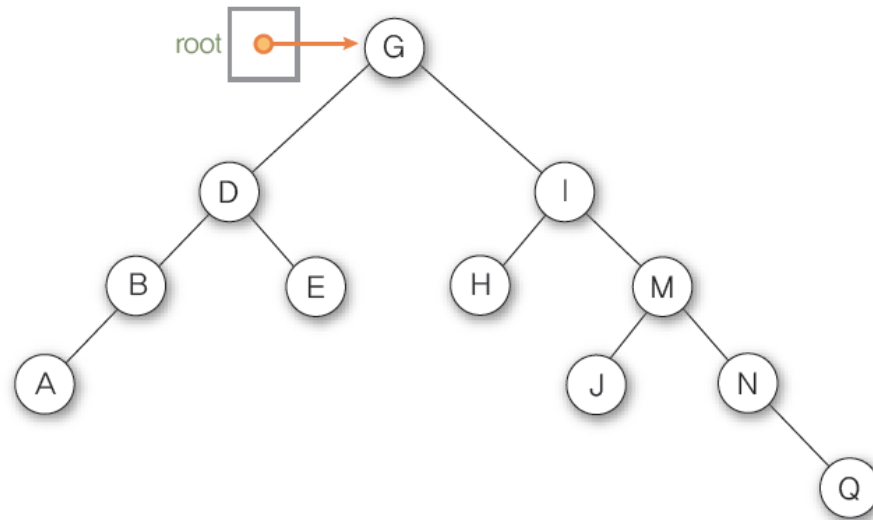


그림 7-43 연산에 사용할 이진 탐색 트리