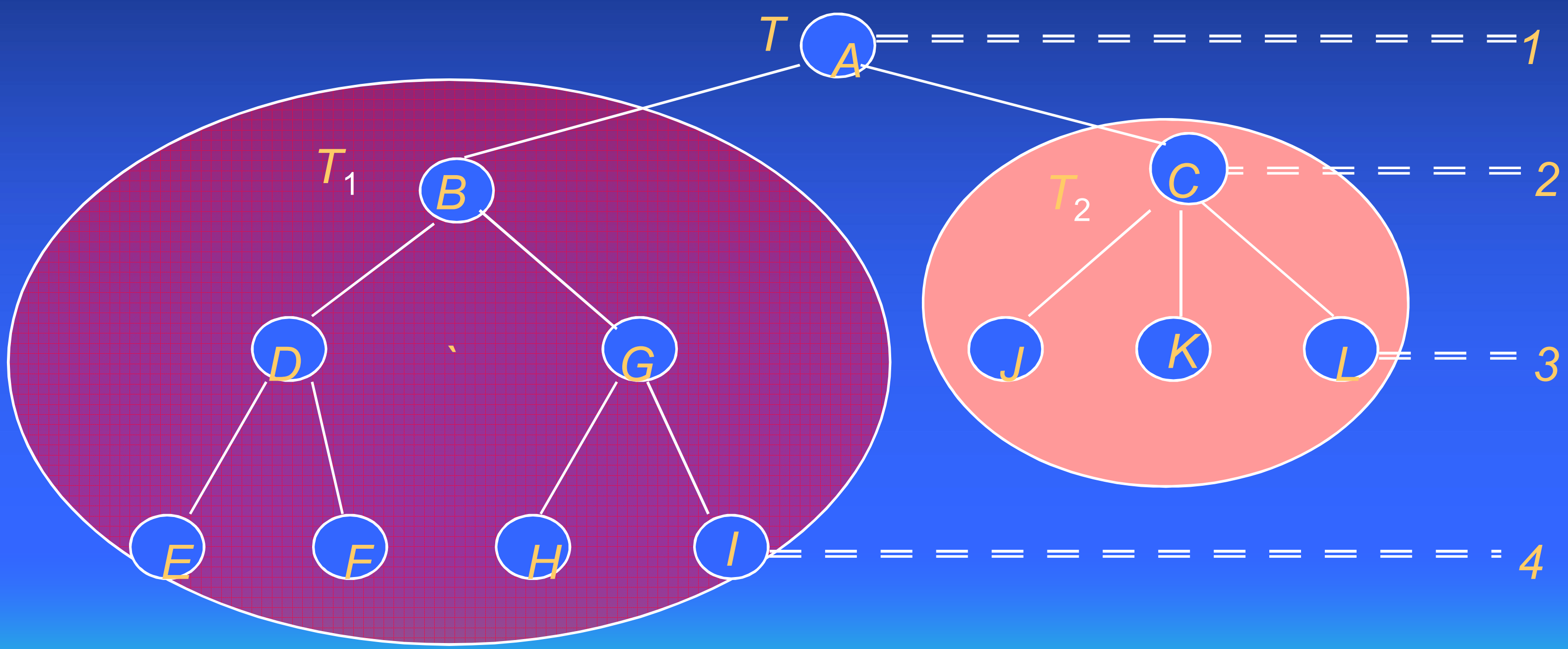
트리(Tree)

1. 트리의 정의 및 용어

- 정의) 트리(tree) : 1개 이상의 노드로 이루어진 유한 집합
  1) 루트(root)라고 하는 노드가 하나 존재
  2) 나머지 노드들은 n≥0개의 분리집합 $T_1, T_2, \ldots, T_n$ 으로 분리.
  3) $T_1, T_2, \ldots, T_n$ 은 각각 하나의 트리이며, 루트의 서브트리(subtree).



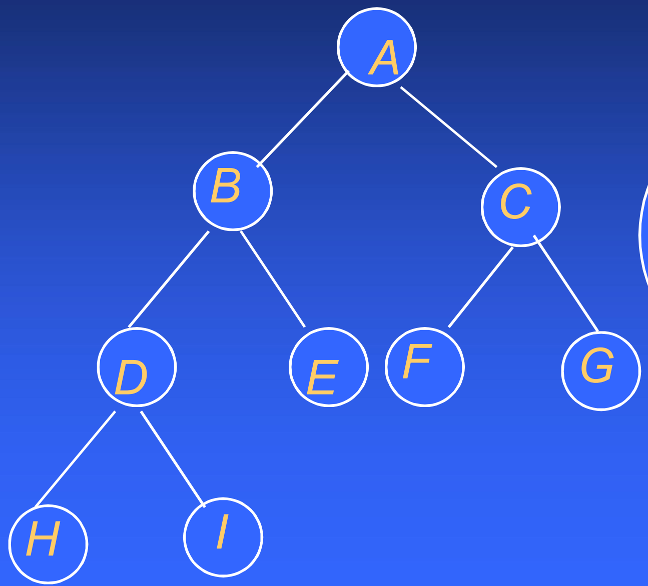
트리의 용어

- 트리의 차수(degree) : 서브트리의 개수
- 부모 노드(parent node) : 상위 레벨에 있으면서 에지로 연결된 노드
- 자식 노드(child node) : 하위 레벨에 있으면서 에지로 연결된 노드
- 터미널 노드(terminal node) 또는 잎(leaf) : 자식 노드가 없는 노드
- 비 터미널 노드(non-terminal node) : 터미널 노드가 아닌 노드
- 형제/자매 노드(siblings) : 같은 레벨에 있으면서 같은 부모를 가진 노드
- 선조(ancestors) : 루트로부터 그 노드까지의 연결된 노드
- 후손(descendents) : 그 노드의 서브트리내의 모든 노드
- 트리의 높이(height)/길이(depth) : 트리에 속한 노드의 최대 레벨

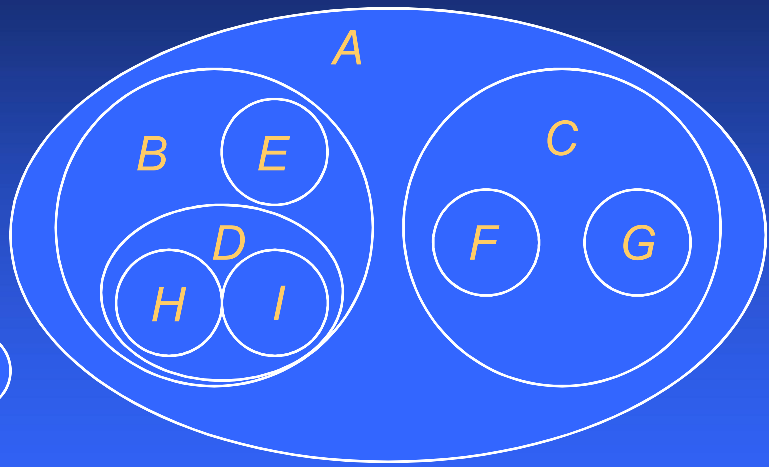
2. 트리의 표현

(1) 리스트표현

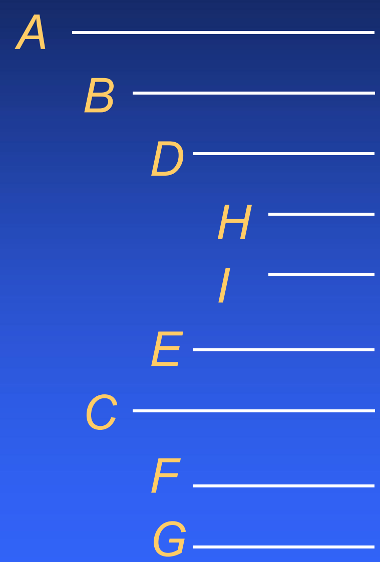
(A(B(D(H,I)E),C(F,G)))



(2)트리표현



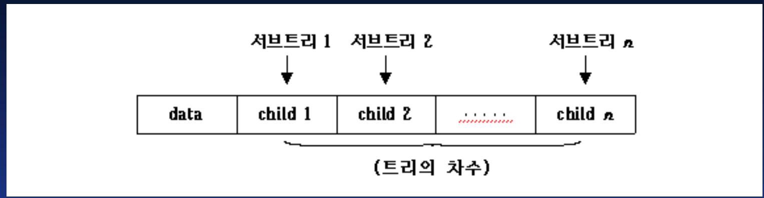
(3)집합표현



(4)들여쓰기표현

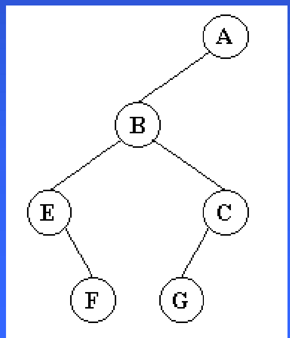
(2) 트리의 표현

- 트리를 연결 리스트로 나타내기 위해서 서브트리 개수만큼의 링크 필드가 필요 => 링크 필드의 메모리를 낭비.



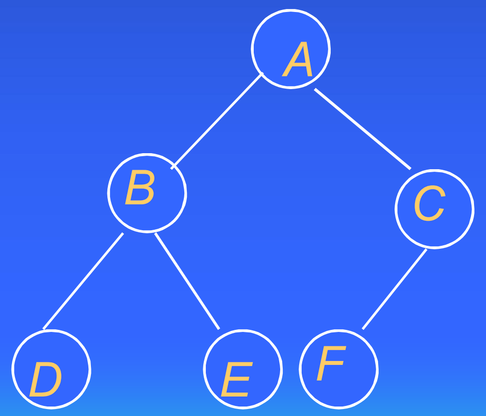
- 트리를 차수가 2인 트리로 표현하여 링크 필드를 절약

<차수가 2인 이진 트리(binary tree)로 변환한 경우>

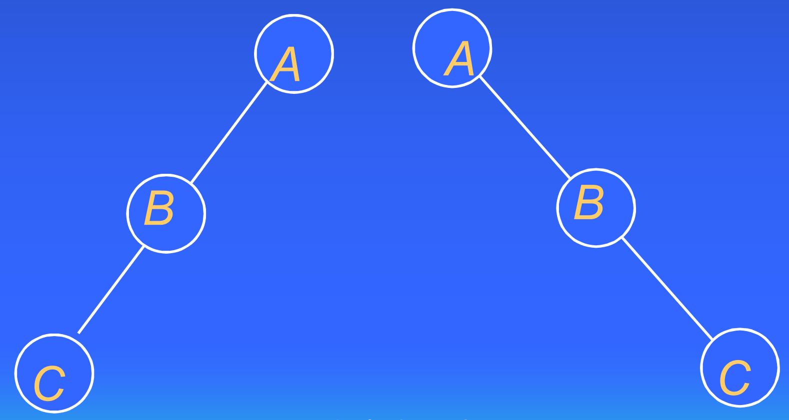


3. 이진 트리

- 정의) 이진 트리 : 공집합이거나 루트와 왼쪽 서브트리, 오른쪽 서브트리라고 부르는 두 개의 분리된 이진 트리로 구성된 노드의 유한집합
- 이진 트리의 종류
  - 완전 이진트리(complete binary tree)
    - 완전이진트리
    - 깊이가 k인 이진 트리에 차례대로 붙인 1부터 n까지의 번호에 노드들이 1대1로 대응하는 트리
  - 사향 이진트리(skewed binary tree)
    - 노드가 한쪽 방향으로만 치우진 이진트리



완전이진트리



사향이진트리

(1) 이진 트리의 특성

① 이진 트리의 레벨 i에서의 최대 노드 수는?

$$2^{i-1} \quad (i \geq 1)$$

깊이가 k인 이진 트리가 가질 수 있는 최대 노드 수는?

$$\sum_{i=1}^{k} 2^{i-1} = 2^{k} - 1$$

② 모든 이진 트리 T에 대해서

$$n_0 = n_2 + 1 \quad (n_0 : \text{터미널 노드 수}, \quad n_2 : \text{차수가 2인 노드 수})$$

③ 깊이가 k인 완전 이진 트리(full Binary Tree)의 노드 수는?

$$2^{k} - 1 \quad (k \geq 0)$$

(2) 이진 트리의 표현

- 배열 또는 연결 리스트를 이용하여 표현

① 배열을 이용한 표현

- ☞ 이진트리의 각 노드들을 자기 위치번호에 맞는 배열의 인덱스 위치에 저장
- ☞ n개의 노드를 가진 완전 이진 트리 (깊이 $= \lfloor \log_2 n+1 \rfloor$)에서 인덱스 i번째 노드인 경우에

i) $i \neq 1$이면 부모(i) $= \lfloor i/2 \rfloor$, $i=1$이면 i는 루트임.

ii) $2i \leq n$이면 왼쪽 자식(i) $= 2i$, 만일 $2i > n$이면 i는 왼쪽 자식을 가질 수 없음.

(예) $i=2$일 때 왼쪽 자식는 $2 \times 2 = 4$

iii) $2i+1 \leq n$이면 오른쪽 자식(i)$= 2i+1$, 만일 $2i > n$이면 i는 오른쪽 자식을 가질 수 없음.

(예) $i=2$일 때 오른쪽 자식은 $2 \times 2+1 = 5$번째에 있음.

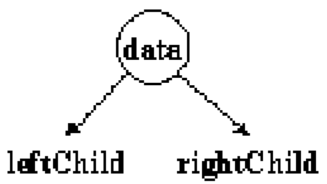
- ☞ 깊이 *k*인 완전 이진 트리는 2^k-1의 노드를 저장할 수 있는 기억장소가 필요.
- ☞ 경사 이진 트리인 경우에는 2^k-1개의 기억장소 중 *k*개만 사용하므로 기억장소가 낭비 => 연결리스트를 이용하여 문제 해결.

(2) 이진 트리의 표현

배열 또는 연결 리스트를 이용하여 표현

② 연결 리스트를 이용한 표현

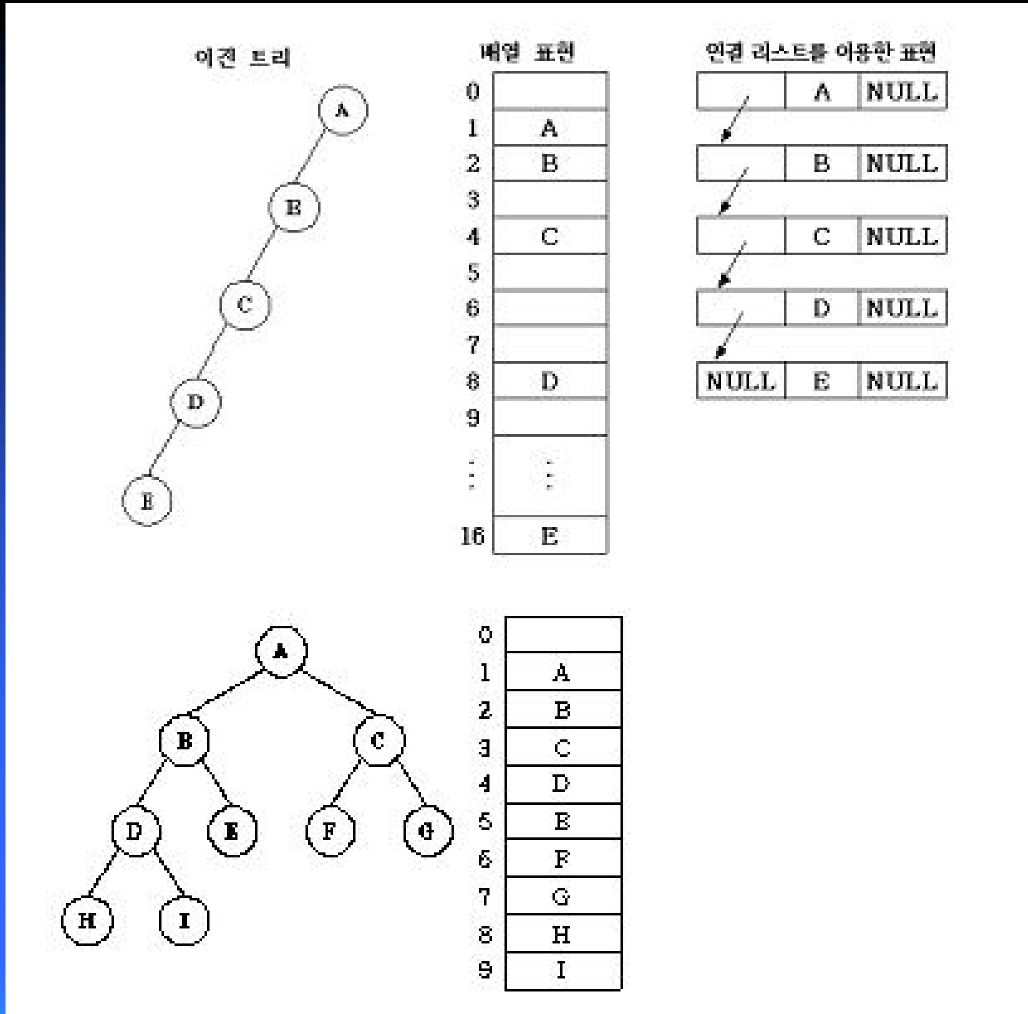
- ☞ 배열을 이용하여 표현할 경우 트리 중간의 노드를 삽입 또는 삭제할 때 노드가 저장된 위치를 변경해야 하는 경우가 발생하는 문제를 해결하기 위하여 연결 리스트를 사용.
- ☞ 각 노드별 왼쪽자식(leftChild)과 오른쪽 자식(rightChild)을 가리키는 두 개의 링크 필드를 정의.



leftChild	data	rightChild

```
typedef   struct   node   *treePtr;
          struct   node{
   int   data;
   treePtr   leftChild, rightChild;
};
```

(2) 이진 트리의 표현



4. 이진 트리 순회(Binary Tree Traversal)

- 중위순회(Inorder Traversal) :

  왼쪽 서브트리 중위순회 → 루트 방문 → 오른쪽 서브트리 중위순회

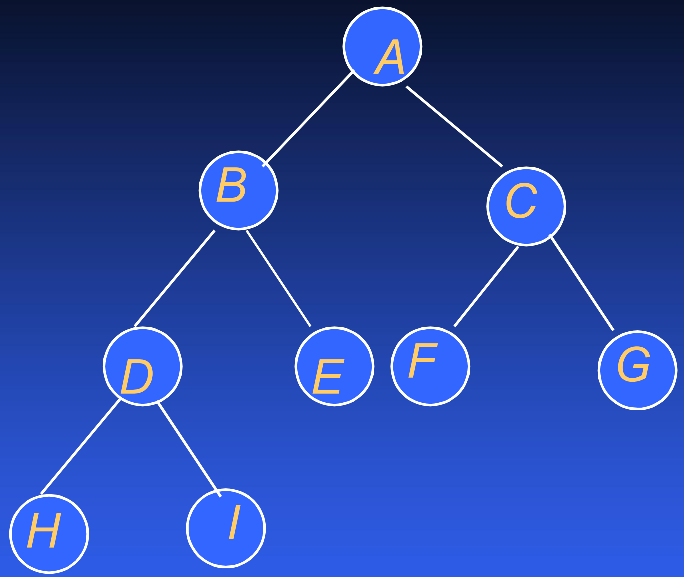
- 후위순회(Postorder Traversal):

  왼쪽 서브트리 후위순회 → 오른쪽 서브트리 후위 순회 → 루트 방문

- 전위순회(Preorder Traversal):

  루트 방문 → 왼쪽 서브트리 전위순회 → 오른쪽 서브트리 전위 순회

4. 이진 트리 순회(Binary Tree Traversal)



전위 운행 순서

ABDHIECFG

중위 운행 순서

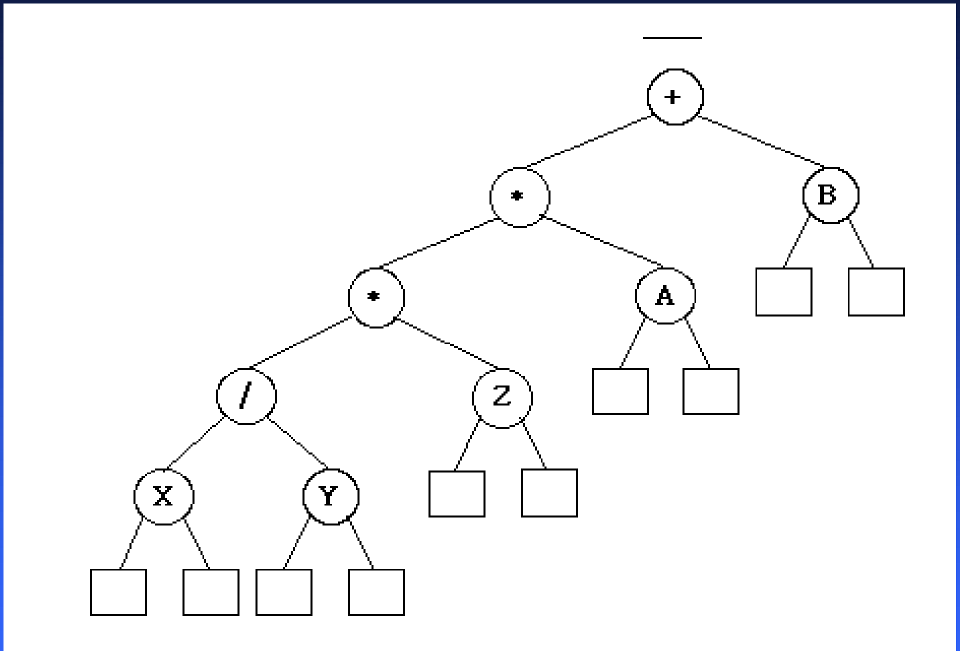
HDIBEAFCG

후위 운행 순서

HIDEBFGCA

이진 트리 순회(Binary Tree Traversal)

- 예) X/Y*Z*A+B 중위 표기(사각형은 NULL 노드)



이진 트리 순회(Binary Tree Traversal)

① 중위순회(Left - Visit - Right)

ⅰ) NULL 노드에 도달할 때까지 Left(왼쪽) 방향으로 이동

ⅱ) NULL 노드에 도착하면 NULL 노드의 부모를 Visit

ⅲ) Right(오른쪽) 방향으로 순회 계속

```
void  inorder(treePtr  ptr)
{
  if(ptr) {
   ① inorder(ptr→leftChild);
   ② printf("%s", ptr→data);
   ③ inorder(ptr→rightChild);
  }
}
```

*순서 : **X/Y*Z*A+B***

4. 이진 트리 순회(Binary Tree Traversal)

② 전위순회(Visit - Left - Right)

ⅰ) 루트부터 노드를 Visit

ⅱ) NULL 노드에 도착할 때가지 왼쪽(Left) 방향으로 이동

ⅲ) NULL 노드에 도착하면 오른쪽(Right) 방향으로 이동

```
void  preorder(treePtr  ptr)
{
  if(ptr) {
    printf("%s", ptr□data);
    preorder(ptr□leftChild);
    preorder(ptr□rightChild);
  }
}
```

*순서 : **+**/XYZAB***

4. 이진 트리 순회(Binary Tree Traversal)

③ 후위순회(Left - Right - Visit)

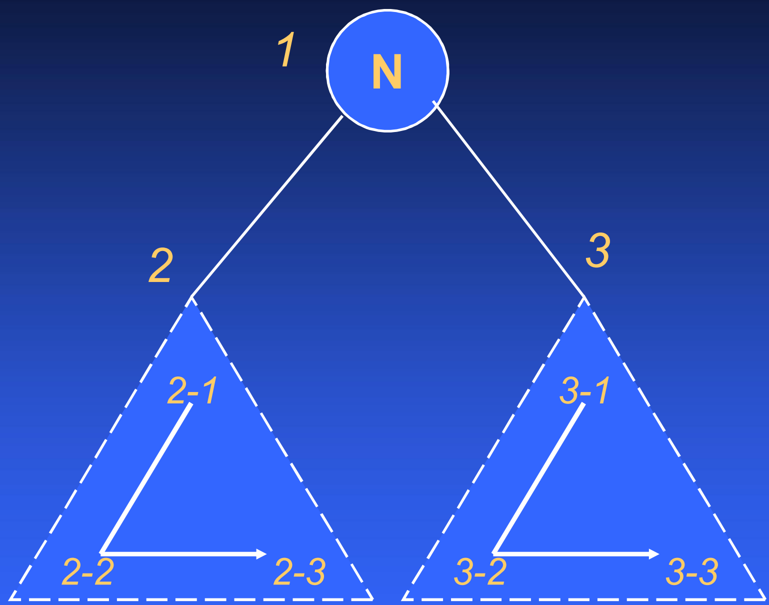
왼쪽 서브트리의 모든 노드들을 출력하고, 오른쪽 서브트리의 모든 노드들을 출력한 후 부모 노드를 출력한다.

```
void  postorder(treePtr  ptr)
{
  if(ptr) {
    postorder(ptr□leftChild);
    postorder(ptr□rightChild);
    printf("%s", ptr□data);
  }
}
```

*순서 : **XY/Z*A*B+***

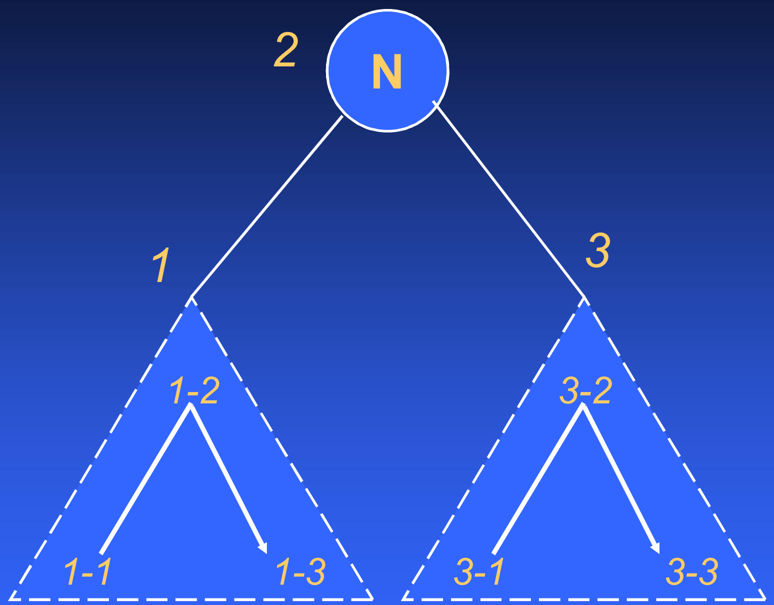
4. 이진 트리 순회(Binary Tree Traversal)

<전위 운행>



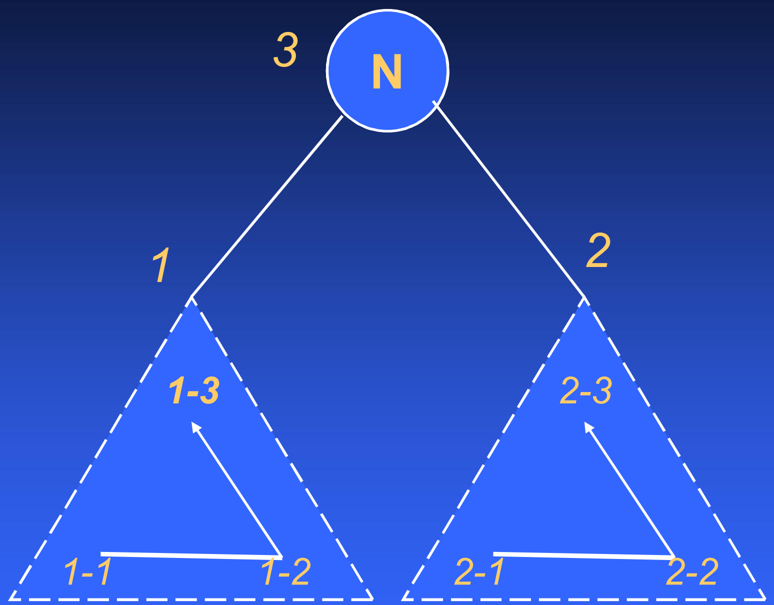
4. 이진 트리 순회(Binary Tree Traversal)

<중위 운행>



4. 이진 트리 순회(Binary Tree Traversal)

<후위 운행>



5. 이진 탐색 트리(Binary Search Tree)

정의) 이진 탐색 트리

① 공백(empty)이거나

② 공백이 아니면 다음 성질을 만족한다.

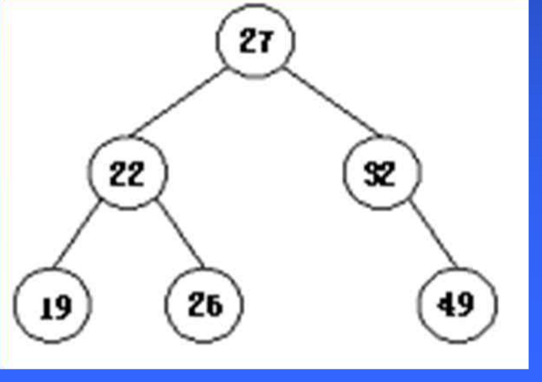
i) 모든 원소는 key를 가지며, key는 유일한 값을 가진다.

ii) 왼쪽 서브 트리에 있는 key들은 루트의 key 보다 작다.

iii) 오른쪽 서브 트리에 있는 키들은 루트의 key 보다 크다.

iv) 왼쪽과 오른쪽 서브 트리도 이진 탐색 트리이다.

예)



5. 이진 탐색 트리(Binary Search Tree)

- 이진 탐색 트리를 이용한 연산

(1) 탐색(Search)

key 값이 k인 노드를 탐색하기 위한 알고리즘은 다음과 같다.

ⅰ) if 루트 == NULL return NULL;
ⅱ) else
    if 루트 key == k return 루트;
    else if 루트 key < k
        오른쪽 서브트리 탐색
    else 왼쪽 서브트리 탐색

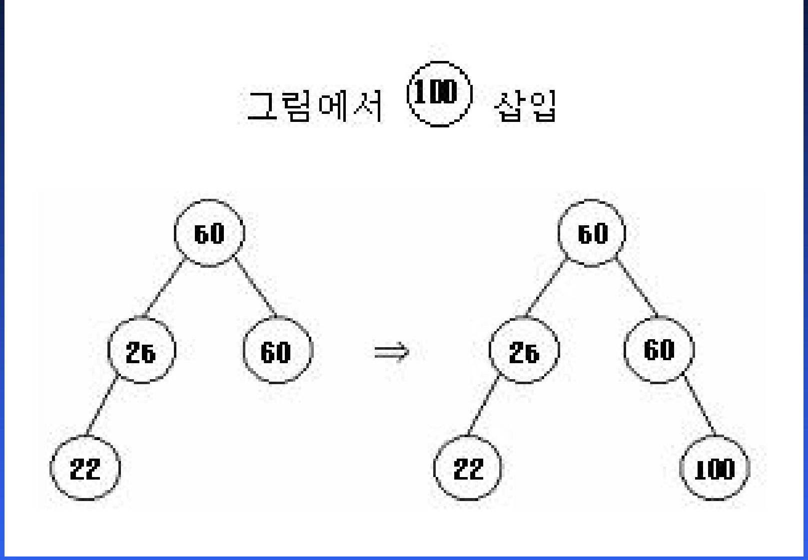
(2) 삽입(Insert)

key 값이 k인 노드를 삽입하는 알고리즘은 다음과 같다.

ⅰ) k가 있는지 탐색한 후
ⅱ) 탐색이 종료된 시점에 삽입

5. 이진 탐색 트리(Binary Search Tree)

이진 탐색 트리의 노드 삽입 과정)



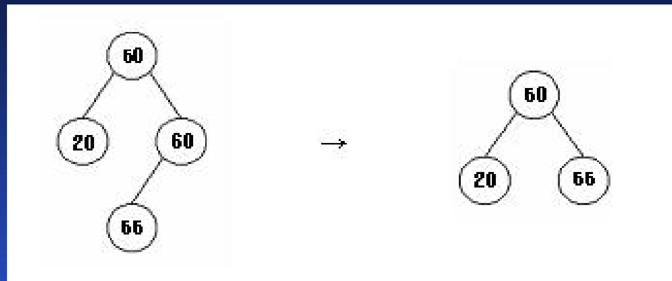
5. 이진 탐색 트리(Binary Search Tree)

(3) 노드 삭제(Delete)

key 값이 k인 노드를 탐색하기 위한 알고리즘은 다음과 같다.

ⅰ) 삭제하고자 하는 노드가 터미널노드 일 때 → NULL로 만들고 반환(return)

ⅱ) 삭제하고자 하는 노드가 하나의 자식만을 가지는 비단말노드(예: 60) 일 때



ⅲ) 삭제하고자 하는 노드가 두 개의 자식을 가지는 비단말노드 일 때

→ 왼쪽 서브트리에서 가장 크거나, 오른쪽 서브트리에서 가장 작은 것을 대체

예) '70'을 삭제하고자 한다면,

①'65'를'70'이 있던 자리로 이동하고

②'62'를'65'가 있던 자리로 이동한다.