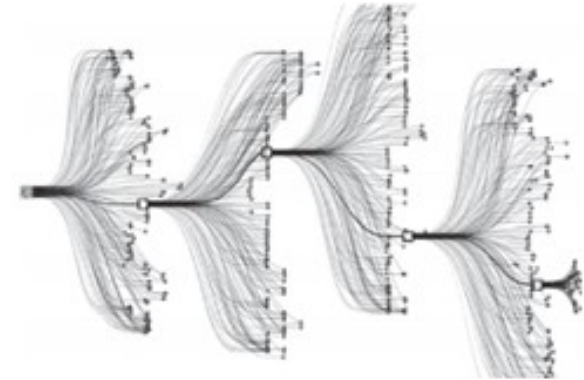
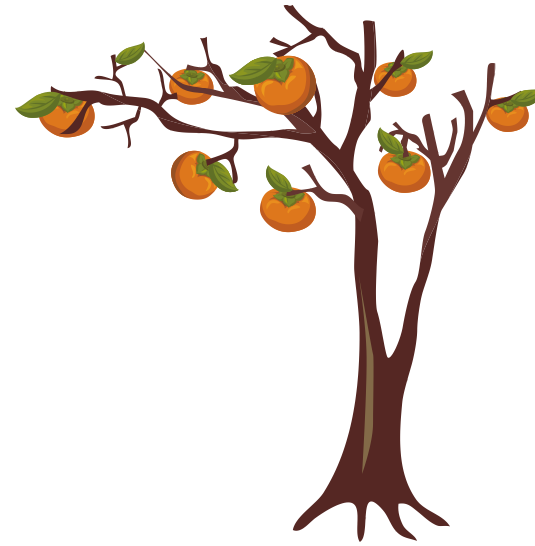
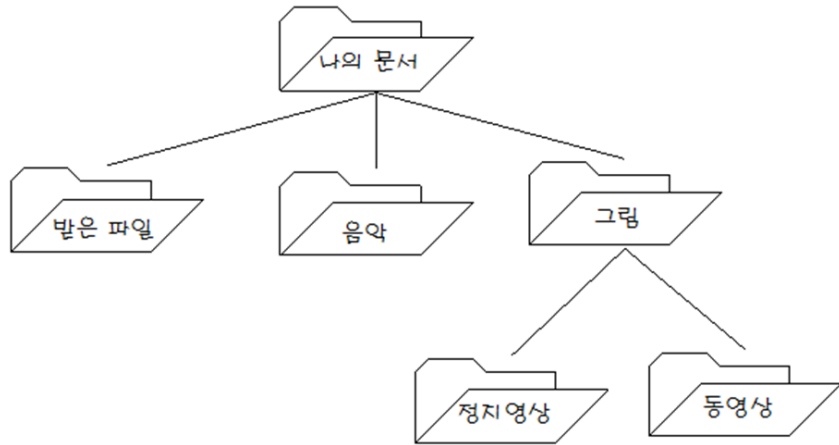
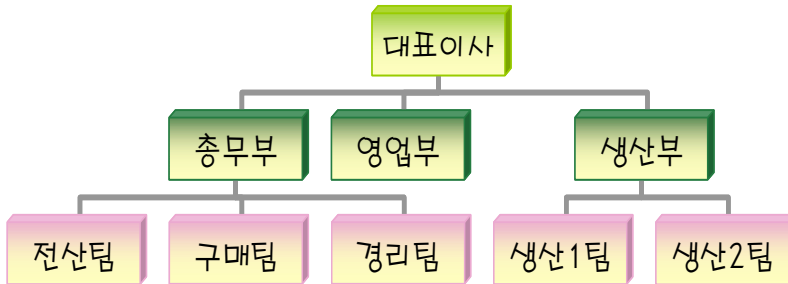




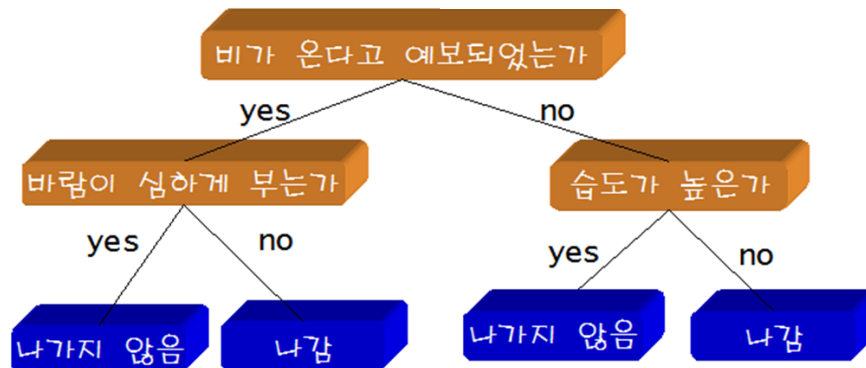
6장. 트리

- 트리의 의미
- 트리의 표현
- 트리 순회
- 트리의 응용

학습목표



- ▶ 트리의 의미
- ▶ 트리의 표현
- ▶ 트리 순회
- ▶ 트리의 응용



❖ 한 개 이상의 노드로 구성되며, 루트와 subTree의 계층으로 구성되는 자료구조.

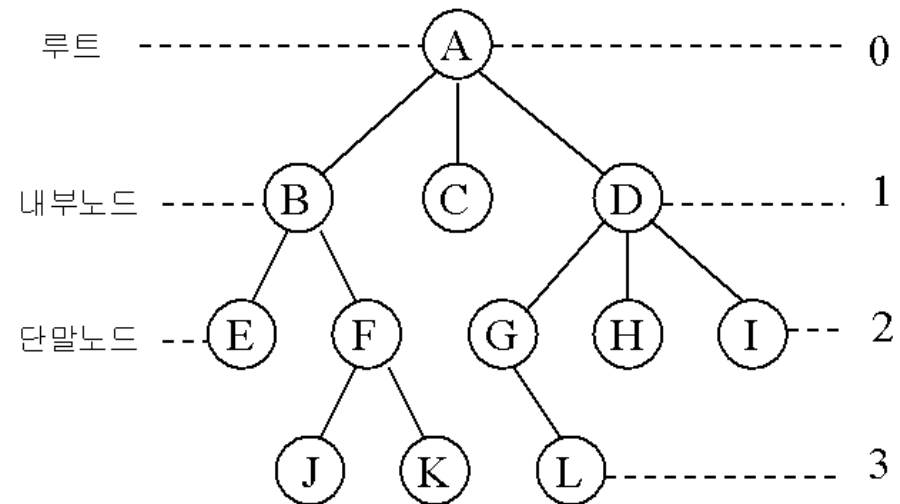
- 그래프에 제한을 가하여 구성한 비순환적 그래프로서, 루트를 최고(最高)로 하는 계층형 자료구조 (hierarchical data structure).
- 사이클을 포함하지 않는 연결 그래프로서 임의의 마디(node)에서 다른 마디로의 경로가 하나 밖에 없는 그래프.
- 트리의 순환적 정의
 - 하나의 노드는 그 자체로 트리이면서 해당 트리의 루트가 됨
 - 만일 n 이 노드이고 $T_1, T_2, \dots, T_k$ 가 $n_1, n_2, \dots, n_k$ 를 루트로 갖는 트리라고 할 때
 - n 을 부모로 $n_1, n_2, \dots, n_k$ 를 연결 – 새로운 트리 생성
 - n 은 루트(root)
 - $T_1, T_2, \dots, T_k$ 는 루트 n 의 서브트리(subtree)
 - 노드 $n_1, n_2, \dots, n_k$ 는 노드 n 의 자식들(children)

트리의 기본 개념(이론적 배경)

레벨

■ 관련 용어

- 노드(node), 간선(link, edge)
- 루트(root)
- 자식(children, 노드 x의 서브 트리 루트들)
- 부모(parent, 노드 x의 상위 노드)
- 형제(siblings, 한 부모의 자식들)
- 조상(ancestor) : 직계조상
- 자손(descendants) : 모든 자손
- 노드의 차수(degree, 한 노드가 가지고 있는 서브 트리의 수)
- 트리의 차수(그 트리에 있는 노드의 최대차수)
- 단말노드(terminal 노드, leaf, 차수가 0인 노드)
- 비단말노드(nonterminal, 차수가 1 이상인 노드, 내부노드)
- 노드의 레벨(level, 루트를 레벨 0으로 할 때, 한 노드가 레벨 l에 속하면, 그 자식들은 레벨 l+1에 속함)
- 트리의 높이(height) 또는 깊이(depth) : 그 트리의 최대 레벨 + 1
- 노드의 레벨순서(level order, 트리를 구성하고 있는 노드들에 순서를 부여한 것으로 : 위에서 아래로, 같은 레벨 안에서는 왼쪽에서 오른쪽으로 차례로 순서를 매긴 것),
- 포리스트(forest, 분리된 트리의 집합)



트리의 표현

어떤 정보들이 표현되어야 할까?

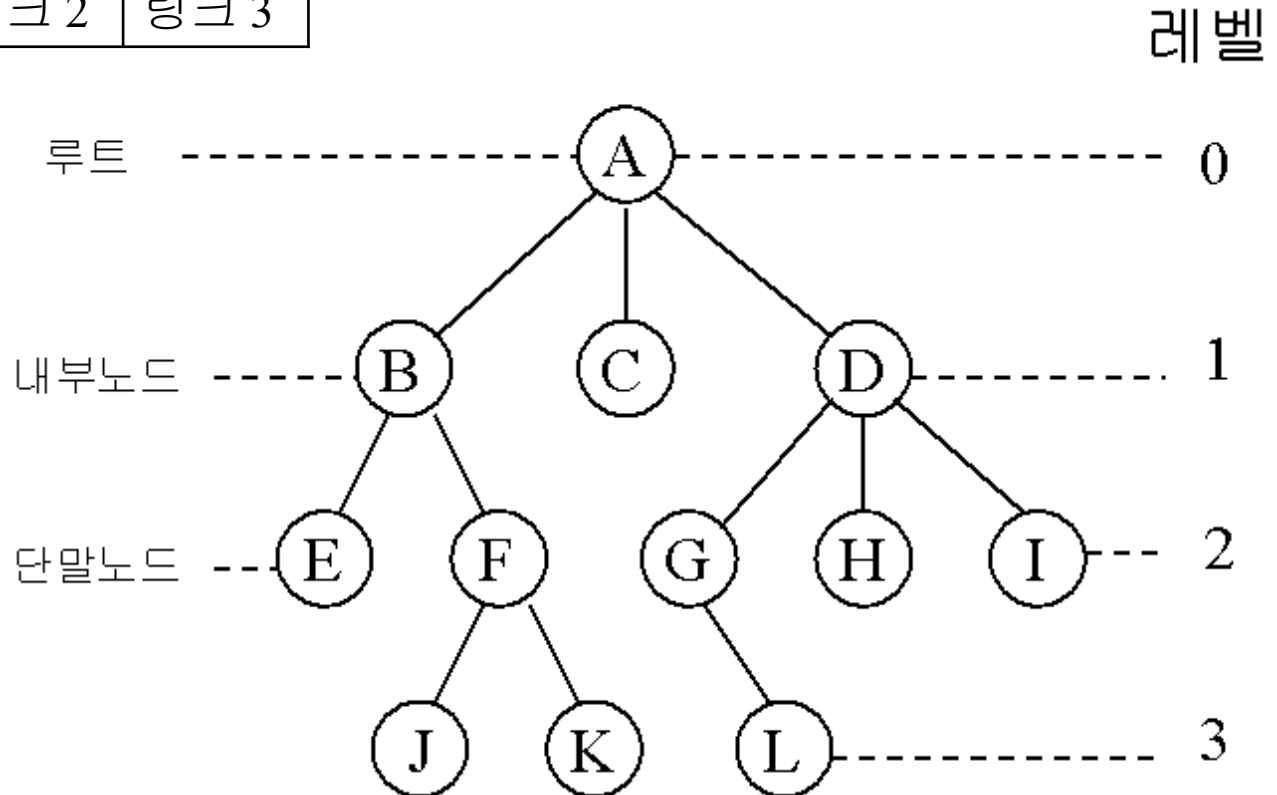
❖ 리스트 표현 (문자 배열 이용 표현)

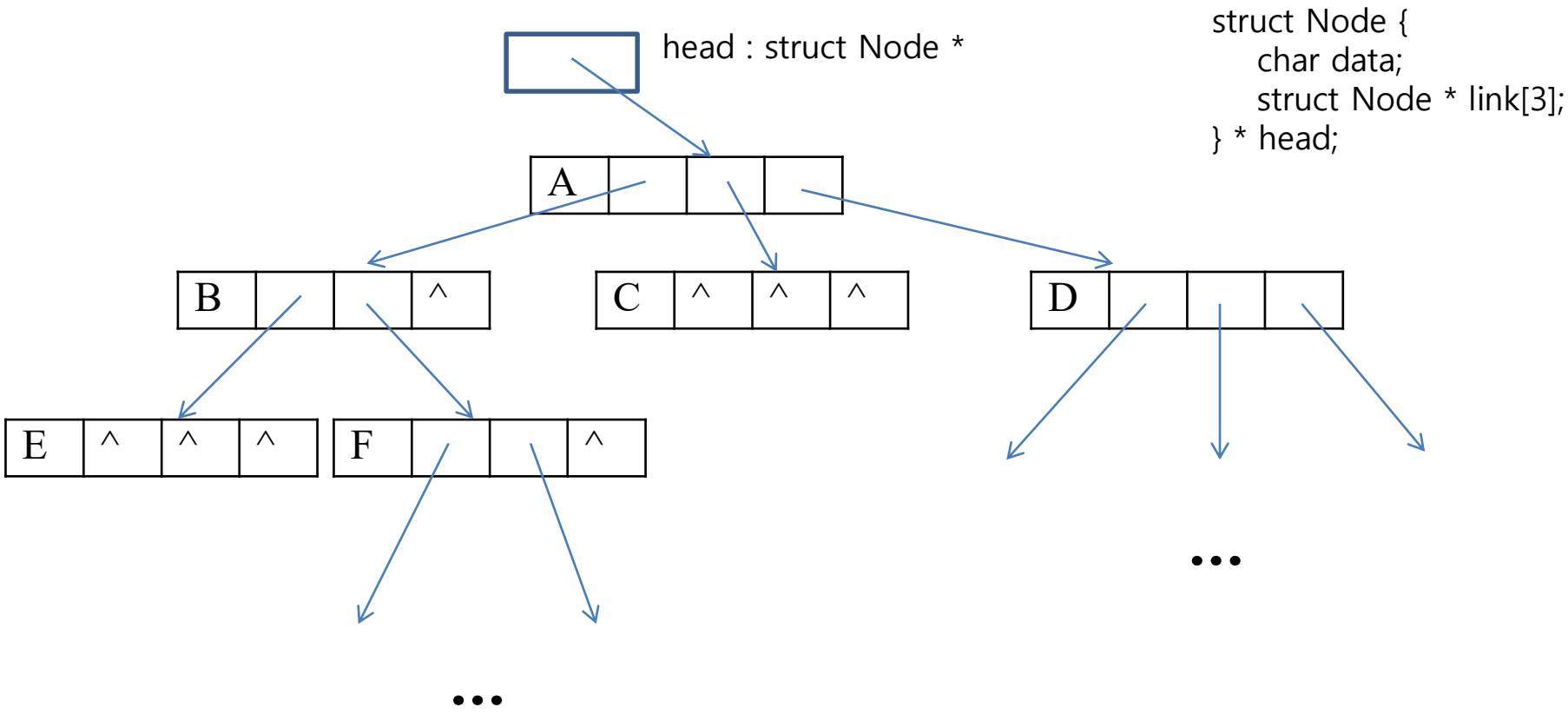
(A(B(E, F(J, K)), C, D(G(L), H, I)))

❖ 연결 리스트 표현

데이터	링크 1	링크 2	링크 3
-----	------	------	------

Q. 연결리스트로 표현해 보세요.





```
struct Node {
    char data;
    struct Node * link[3];
} * head;
```

*. 효율적인 알고리즘
작성을 위해 노드 구조가 일정한 것이 좋음.
=> 이진 트리

이진 트리(Binary tree)

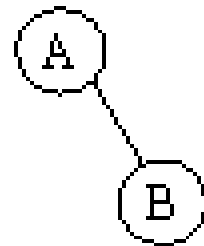
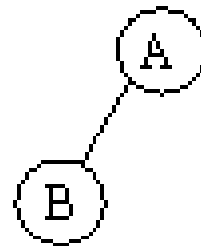
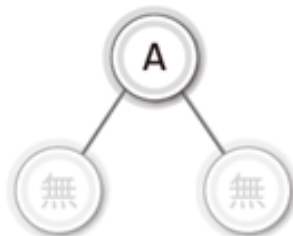
❖ 공집합이거나

❖ 루트와 왼쪽 서브 트리, 오른쪽 서브 트리로 구성된 노드의 유한 집합.

- 일반 트리가 다양한 수의 서브 트리(자녀)를 갖는 것과 달리 서브트리의 수를 2로 제한한 트리. 컴퓨터 응용에서 가장 많이 사용.
- 이진 트리에서는 공백 트리도 tree로 간주하며,
- 왼쪽 서브 트리와 오른쪽 서브 트리를 분명하게 구별함(일반 트리는 위치 개념이 없음).
 - * 용어 추가 :
'왼쪽 서브트리'와 '오른쪽 서브트리' 추가.

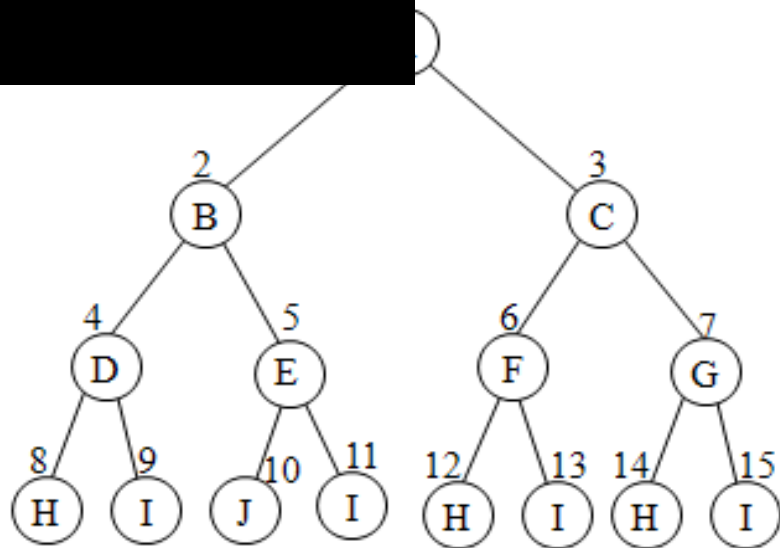


→
달리 표현하면

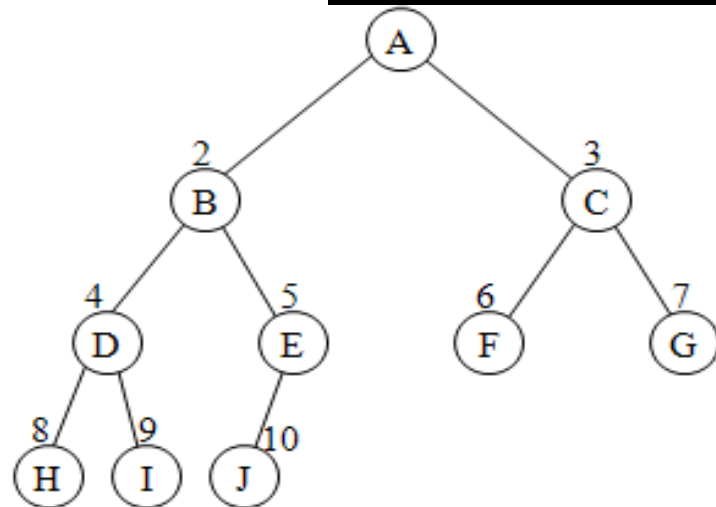


이진 트리 종류

사향(편향, 기울어진) 이진 트리
포화 이진 트리
완전 이진 트리



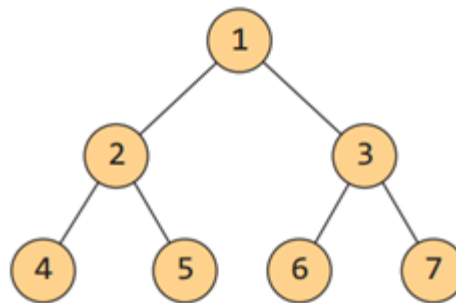
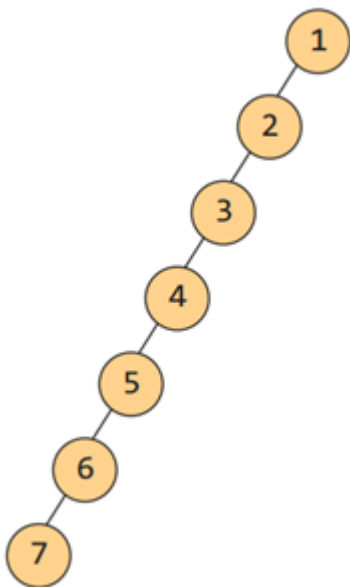
높이가 3인 포화 이진 트리



완전 이진 트리

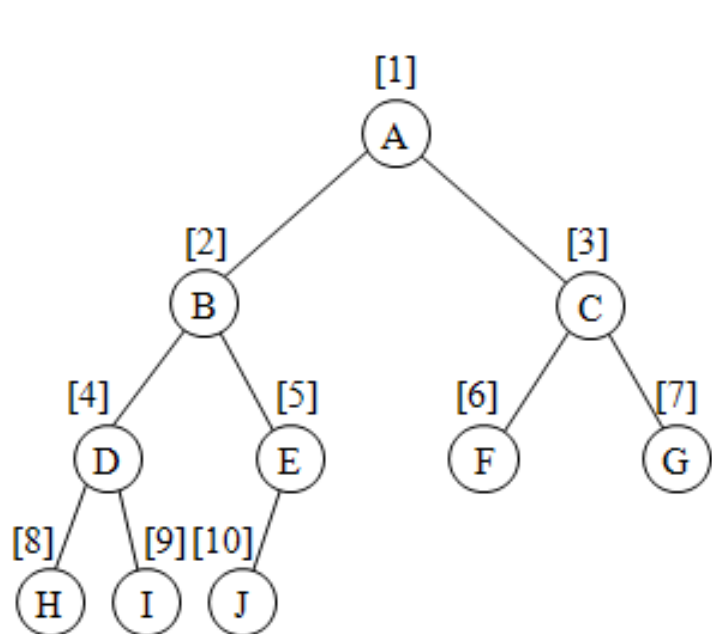
이진 트리의 성질

- ❖ 노드 수가 n 이면 간선 수는 $n-1$
- ❖ 높이가 h 이면 노드 수는 $h \sim 2^h - 1$
- ❖ 노드 수 n 일 때 트리 높이는 $\lceil \log_2 (n+1) \rceil \sim n$

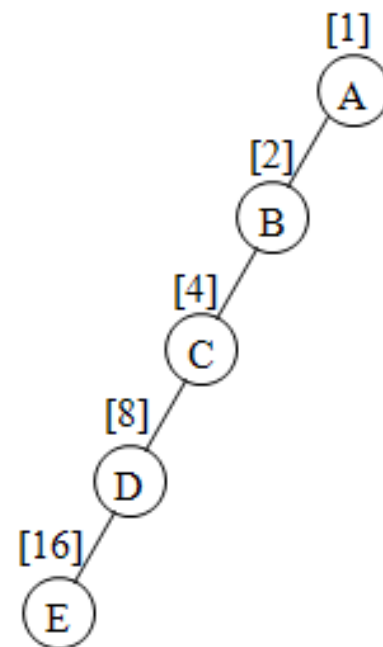


이진 트리의 표현

❖ 배열에 의한 표현



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



[0]	-
[1]	A
[2]	B
[3]	-
[4]	C
[5]	-
[6]	-
[7]	-
[8]	D
.	-
.	...
.	-
[16]	E

목표 노드	인덱스 값	조 건
노드 i의 부모	$\lfloor i/2 \rfloor$	$i > 1$
노드 i의 왼쪽 자식	$2*i$	$2*i \leq n$
노드 i의 오른쪽 자식	$2*i + 1$	$(2*i + 1) \leq n$
루트 노드	1	$0 < n$

*. n : 마지막 노드 첨자

이진 트리의 표현

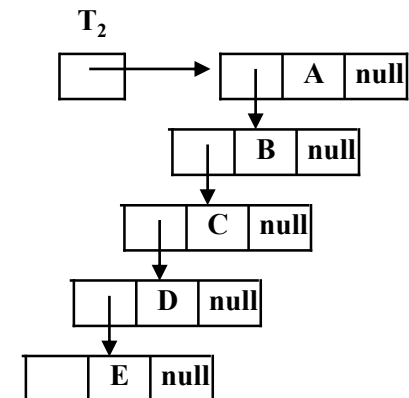
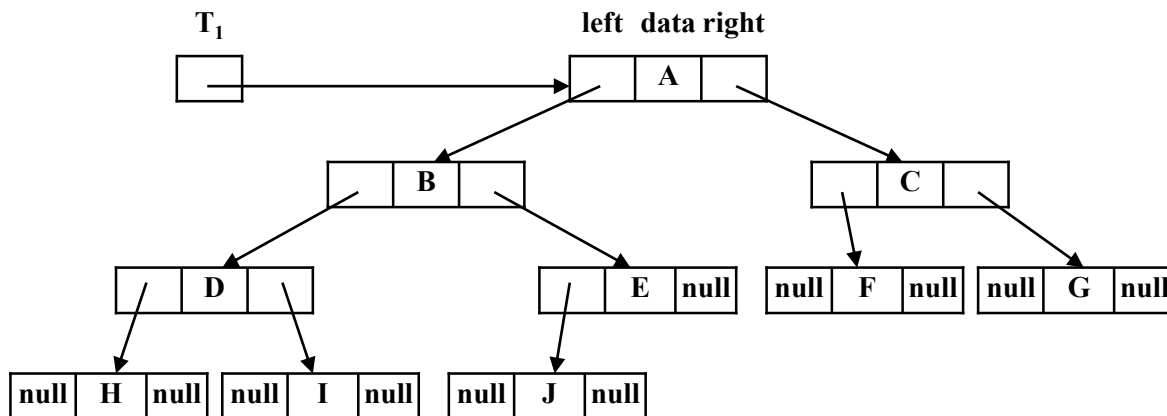
❖ 연결 리스트 표현

- 각 노드를 3개의 필드 left, data, right로 구성

left	data	right
------	------	-------

*. 필요 시 부모를 가리키는 parent 필드 추가

- left와 right : 왼쪽 서브트리와 오른쪽 서브트리를 가리키는 링크
- 완전 이진 트리와 편향 이진 트리의 연결 리스트 표현



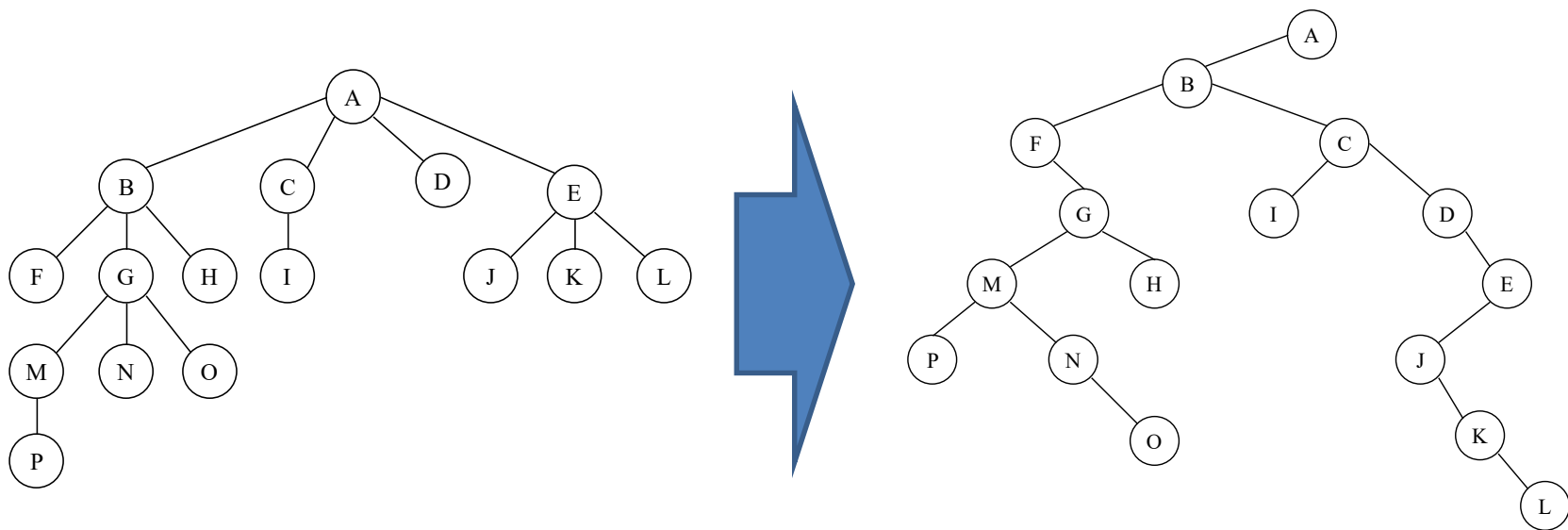
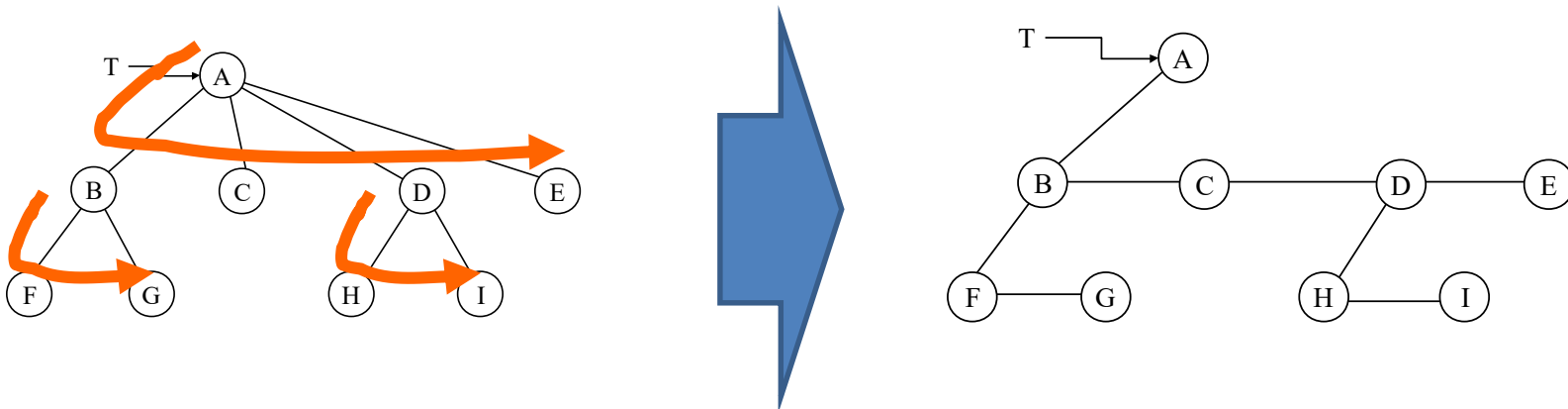
❖ Tree ADT

데이터 : 정점과 간선의 집합
연산

- isEmpty() : 트리가 공백 상태인지 확인한다.
- isLeaf(v) : 정점 v가 단말 노드인지 확인한다.
- degreeNode(v) : 정점 v의 차수를 반환한다.
- getParent(v) : 정점 v의 부모 노드를 반환한다.
- getLeftChild(v) : 정점 v의 왼쪽 자식 노드를 반환한다.
- getRightChild(v) : 정점 v의 오른쪽 자식 노드를 반환한다.
- getSibling(v) : 정점 v의 형제를 반환한다.
- getAncestors(v) : 정점 v의 (직계)조상들을 반환한다.
- getDescendants(v) : 정점 v의 자손들을 반환한다.
- getLevel(v) : 정점 v의 레벨을 반환한다.
- getHeight() : 트리의 키를 반환한다.
- countNode() : 정점의 수를 반환한다. //순회를 이용하여 프로그램 구현
- degreeTree() : 트리의 차수를 반환한다.

일반 트리의 이진 트리화

모든 노드를 부모 노드, 첫째 자식 노드, 그리고 다음 형제 노드 순으로 연결 후 정리



이진 트리의 순회

❖ 전위, 중위, 후위, 레벨 순회(Preorder, Inorder, Postorder, Level Traversal)

■ 순환적(recursive) 전위 순회

- 루트노드(root node)를 방문.
- 왼편 서브트리 전위 순회.
- 오른편 서브트리 전위 순회.

■ 순환적 중위 순회

- 왼편 서브트리 중위 순회.
- 루트노드(root node)를 방문.
- 오른편 서브트리 중위 순회.

■ 순환적 후위 순회 방법

- 왼편 서브트리 후위 순회.
- 오른편 서브트리 후위 순회.
- 루트노드(root node)를 방문.

■ 레벨 순회

큐에 루트를 넣고 큐가 빌 때까지 아래 단계를 반복.

- 큐에서 하나를 꺼내 방문하며, 꺼낸 노드의 자식 노드를 왼편 서브트리, 오른편 서브트리 순으로 각 서브트리의 루트를 큐에 삽입.

*. 순회에서 자식노드 방문 순서는 Left 다음 Right로 고정하였음.

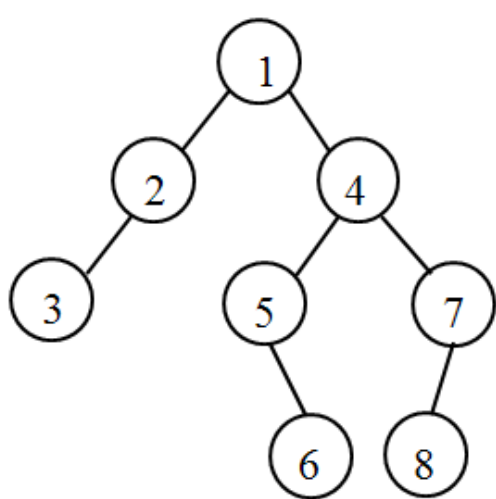
순환적 중위 순회

=====

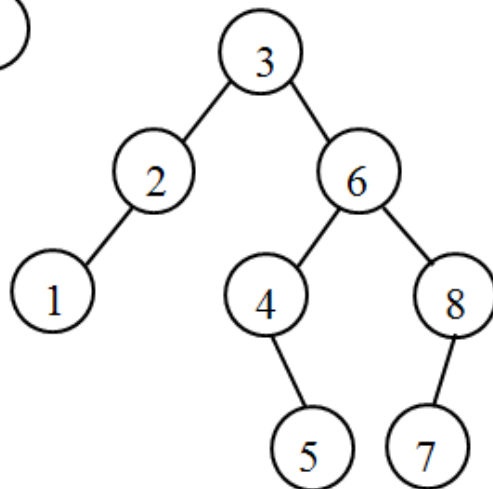
```
inorder(T)
{
    if (T != null) {
        inorder(T.left);
        visit T.data;
        inorder(T.right);
    }
}
```

이진 트리의 순회

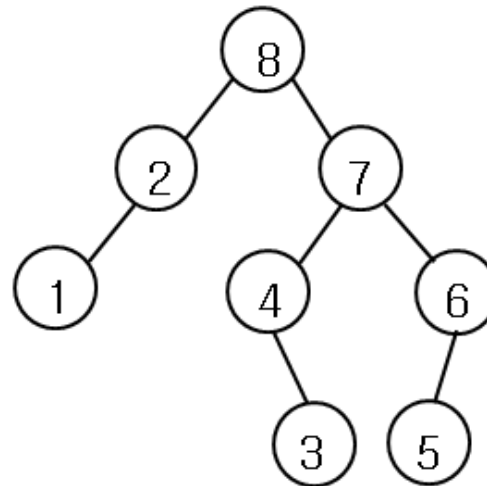
❖ 전위, 중위, 후위, 레벨 순회(Preorder, Inorder, Postorder, Level Traversal)



(a) 전위 순회

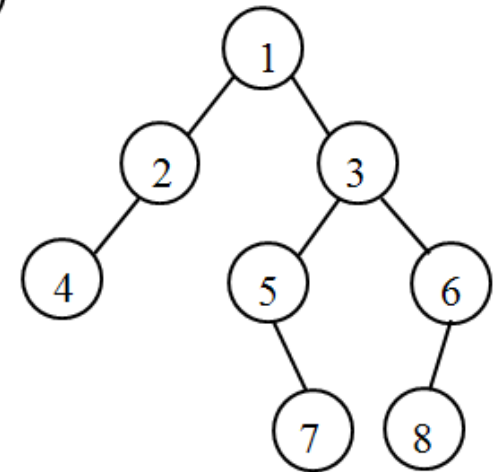


(b) 중위 순회



(c) 후위 순회

Q. 깊이 / 너비 우선 탐색은 무엇과 같을까?



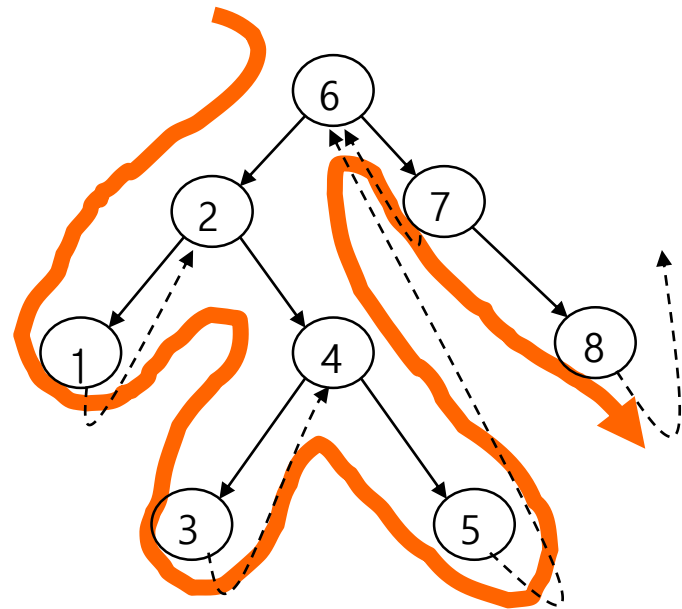
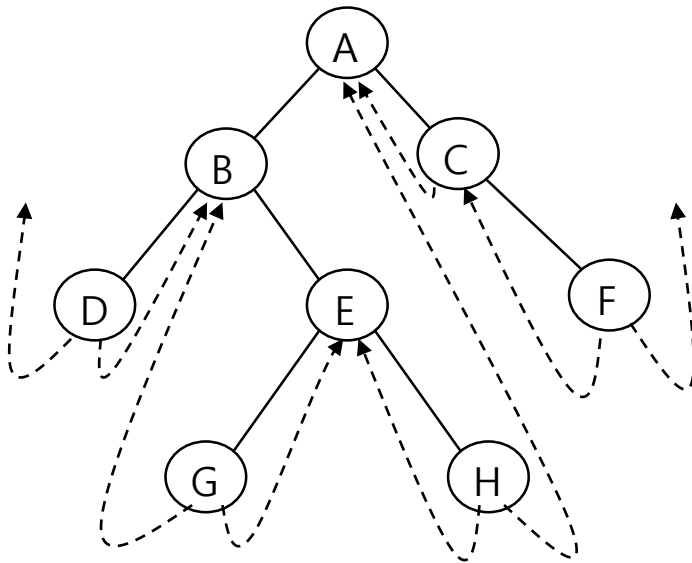
(d) 레벨 순회

스레드이진트리(thread binary tree : TBT)

TBT : 널 링크를 중위 순회에 이용하기 위한 노력이다.

아래 예를 보면

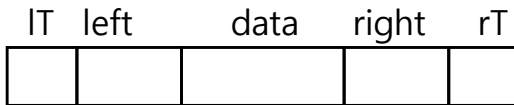
노드 H의 left 필드는 중위 선행자 E에 대한 스레드로 이용하고,
right 필드는 중위 후행자 A에 대한 스레드로 이용한다.



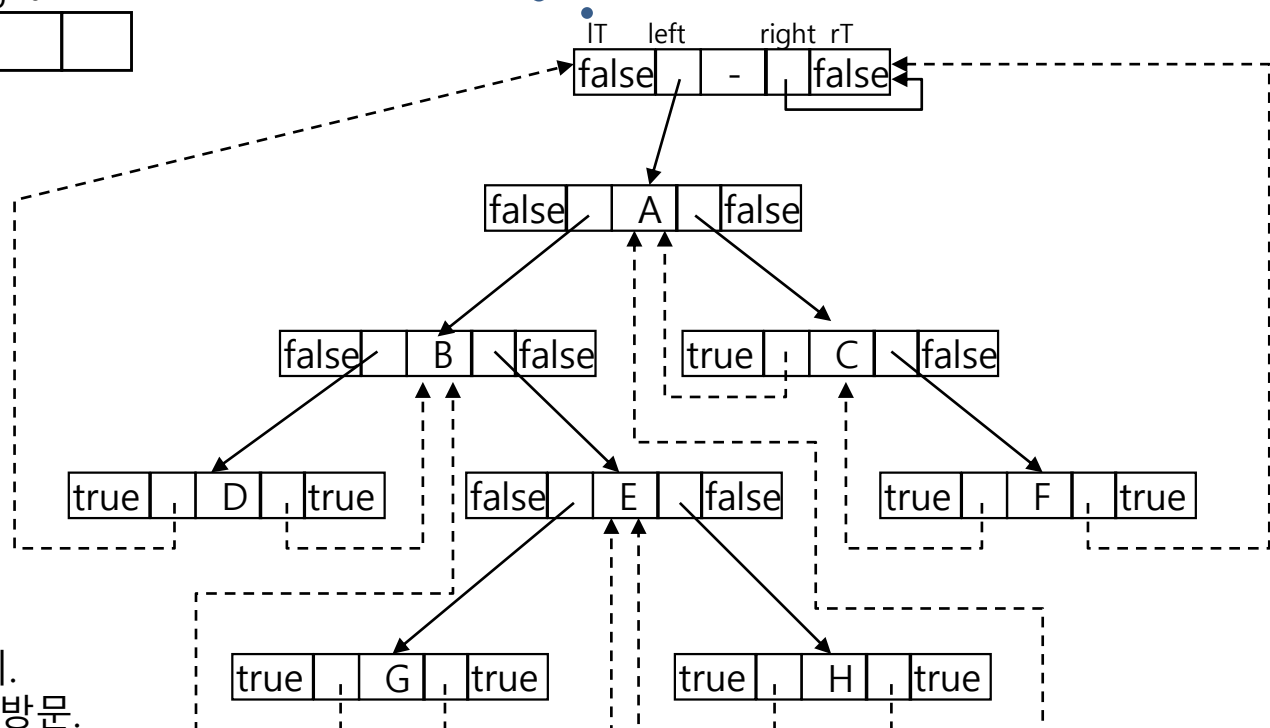
스레드 이진 트리

```
struct TBTNode {
    bool IT;
    struct TBTNode * left;
    char data;
    struct TBTNode * right;
    bool rT;
} header;
```

header : TBTNode

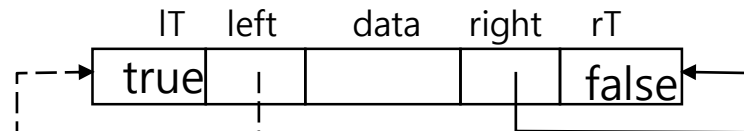


*. IT, rT : left와 right의 스레드와 링크 구별용.



순환적 중위 순회

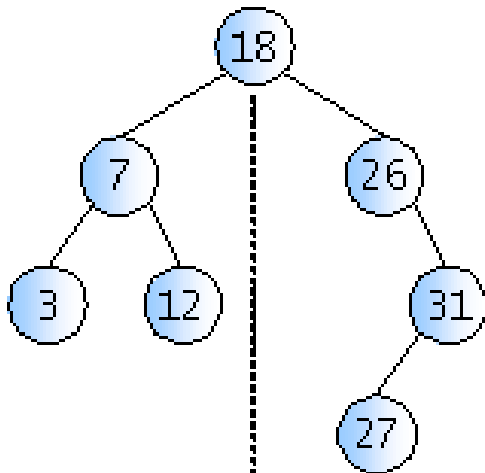
- 왼편 서브트리 중위 순회.
- 루트노드(root node)를 방문.
- 오른편 서브트리 중위 순회.



이진 탐색 트리(Binary Search Tree)

❖ 탐색작업을 효율적으로 하기 위한 자료구조

- $\text{key}(\text{왼쪽서브트리}) < \text{key}(\text{노드}) < \text{key}(\text{오른쪽서브트리})$ 형태로 구성
 - 등호 관계는 의도에 따라 조정 필요
- 이진 탐색을 중위 순회하면 오름차순으로 정렬된 값을 얻을 수 있다.

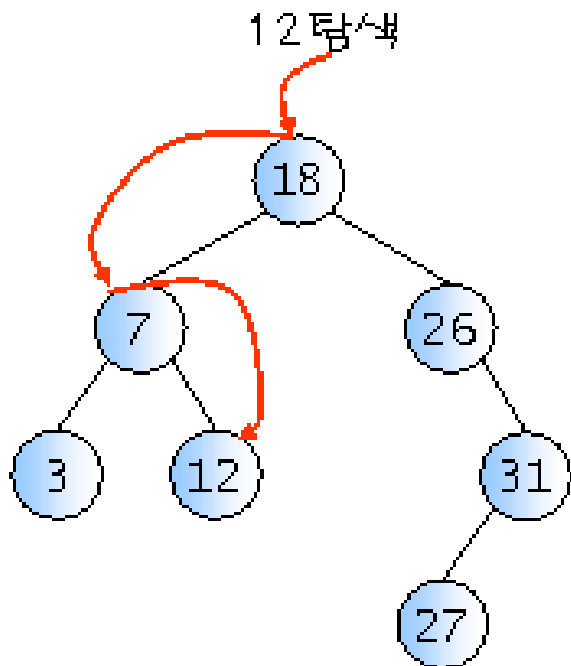


루트보다 작은 값

루트보다 큰 값

이진 탐색 트리에서 탐색 연산

- 1) 방문한 노드 값이 탐색값과 같으면 탐색이 성공적으로 끝난다.
- 2) 탐색 값이 방문 노드 값보다 작으면, 이 노드의 왼쪽 자식을 기준으로 탐색한다. 아니면(크면) 탐색은 이 노드의 오른쪽 자식을 기준으로 다시 탐색한다.

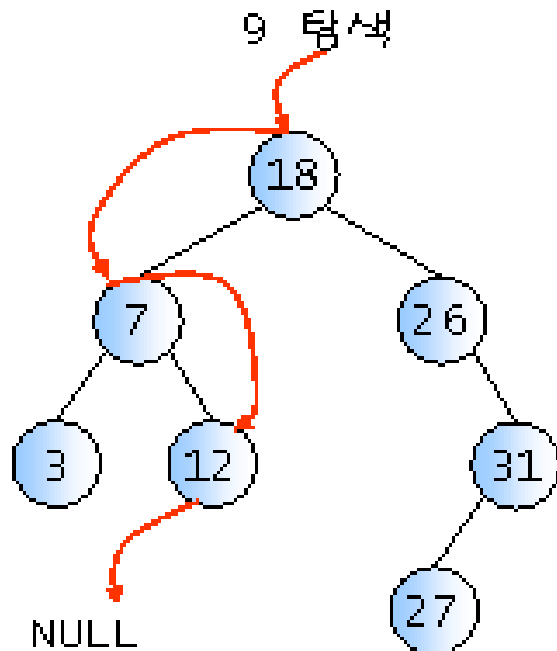


*. 순환적 호출을 통해 쉽게 탐색할 수 있다.

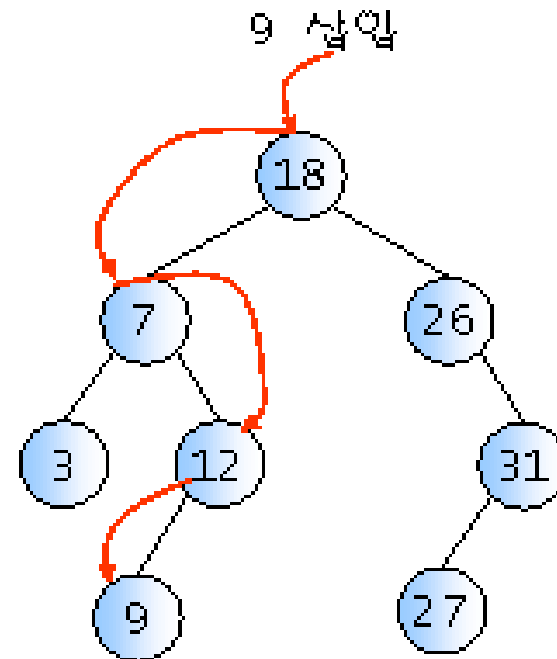
```
search(x, k)
{
    if ( x == NULL)
        then return NULL;
    if ( k == x->key )
        then return x;
    else if ( k < x->key )
        then return search(x->left, k);
    else return search(x->right, k);
}
```

이진탐색 트리에서 삽입 연산

- ❖ 이진 탐색 트리에 원소를 삽입하기 위해서는 먼저 탐색을 수행하는 것이 필요
- ❖ 탐색에 실패한 위치가 바로 새로운 노드를 삽입하는 위치



(a) 탐색을 먼저 수행

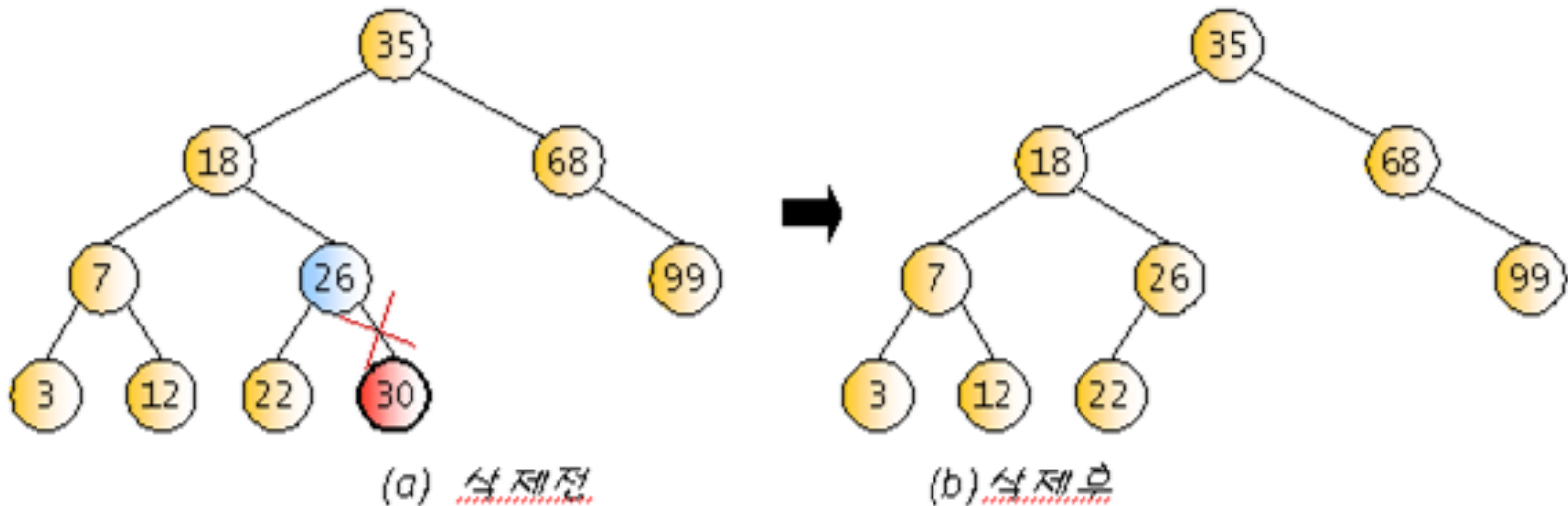


(b) 탐색이 실패한 위치에 9를 삽입

이진탐색 트리에서 삭제연산

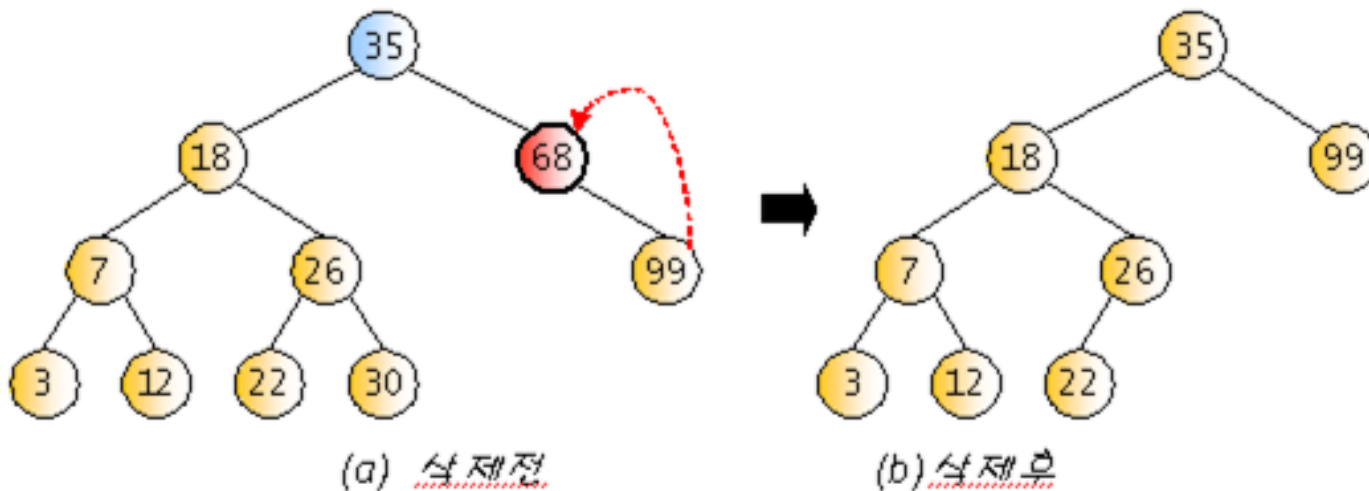
❖ 3가지의 경우가 존재

- Case1 : 삭제하려는 노드가 단말 노드일 경우
 - Case 1: 단말 노드이므로 삭제할 노드의 부모노드를 찾아서 연결을 끊고 노드를 제거한다.

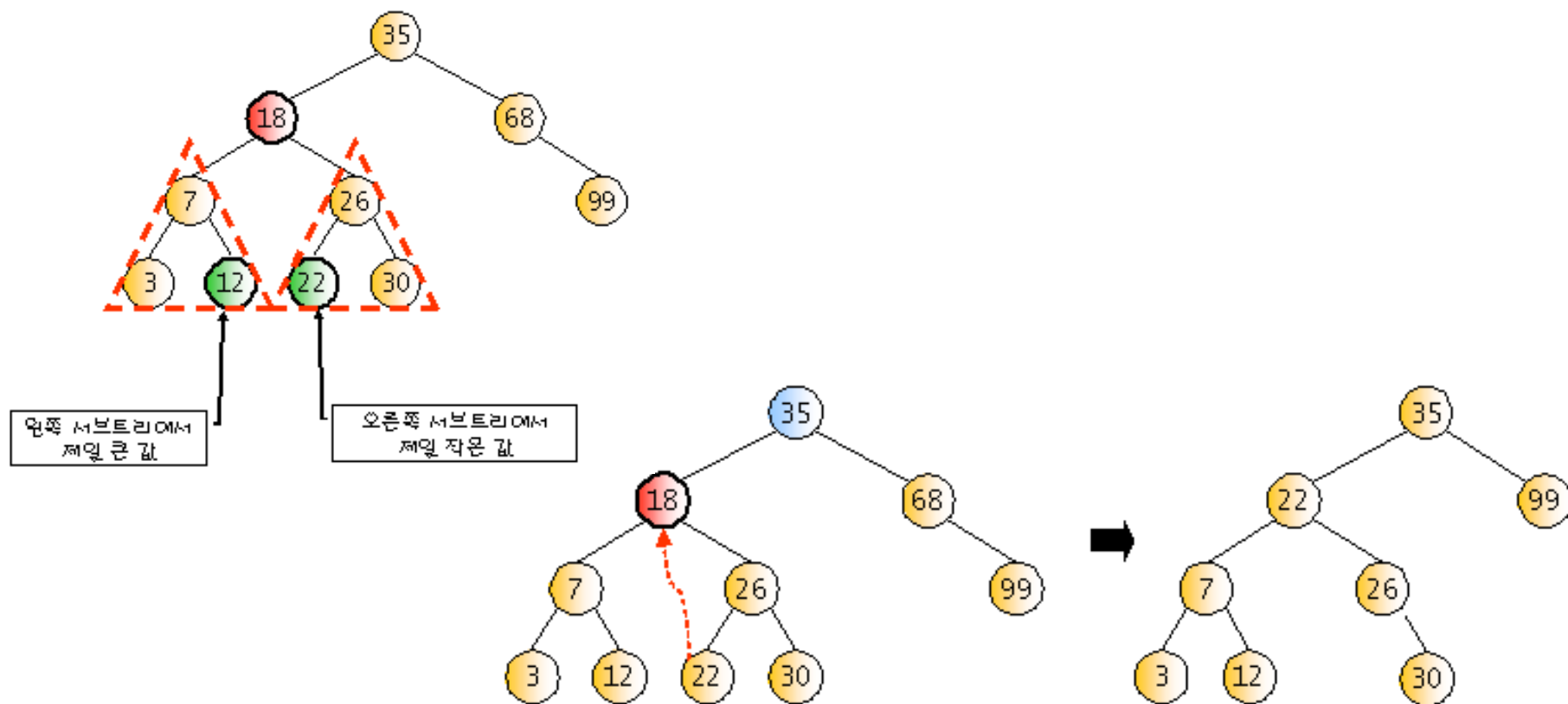


이진탐색트리에서 삭제연산

- Case2 : 삭제하려는 노드가 왼쪽이나 오른쪽 서브 트리 중 하나만 가지고 있는 경우
 - 그 노드는 삭제하고 서브 트리는 부모 노드에 붙여준다.



- Case3 : 삭제하려는 노드가 왼쪽과 오른쪽 서브 트리를 모두 가지고 있는 경우
 - 왼쪽 서브 트리에서 가장 큰 값으로 대체하거나, 오른쪽 서브 트리에서 가장 작은 값으로 대체한다.



우선순위 큐(Priority Queue)와 힙(Heap)

❖ '우선순위 큐'란? : 들어간 순서에 상관없이 우선순위가 높은 자료가 먼저 추출되도록 구성한 자료구조.

- 우선순위 비교기준은 프로그래머가 결정.

❖ 구현 :

- 배열 기반으로 구현.

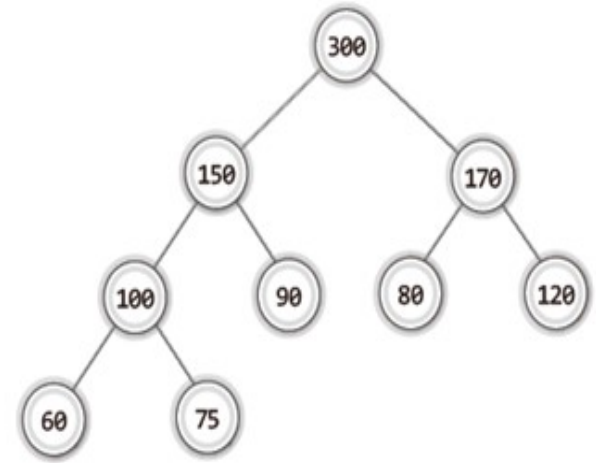


- 연결 리스트 기반으로 구현

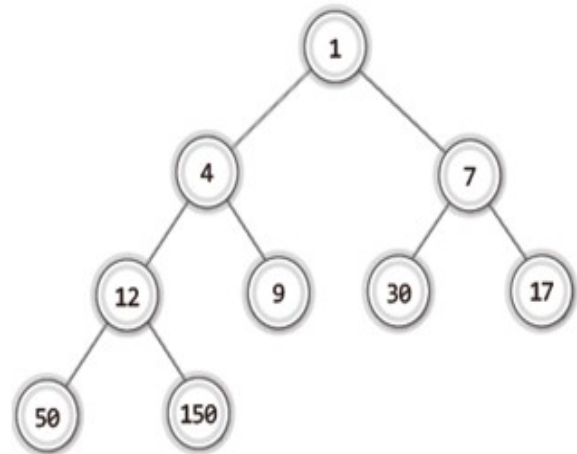


❖ 힙(heap) 트리를 이용한 우선순위 큐.

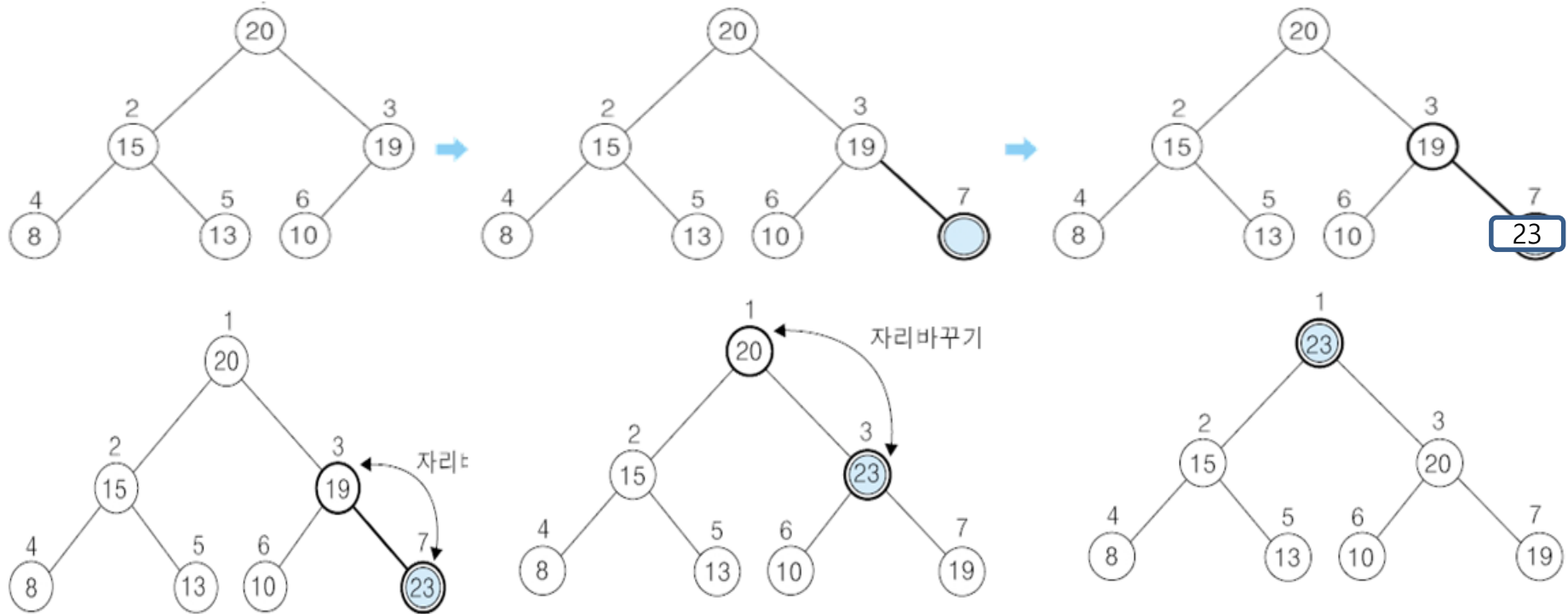
- 힙이란 자식과 부모 노드 관계만을 고려하여 구성한 ‘완전이진트리’
- 종류
 - 최대 힙(max heap) : 부모노드 $\geq$ 자식노드



- 최소 힙(min heap) : 부모노드 $\leq$ 자식노드



■ 힙의 데이터 삽입 (Max Heap으로 예를 보임.)



<< 힙 트리의 삽입 >>

- 1) '완전이진트리'를 유지하며 자료 삽입
- 2) 새로 자료가 삽입된 노드를 중심으로
 while (부모 노드와 관계가 힙 구조 아닌 동안) {
 힙 구조가 완성될 때까지 상향(上向)하며 힙을 구성
 }

■ 힙의 데이터 삭제

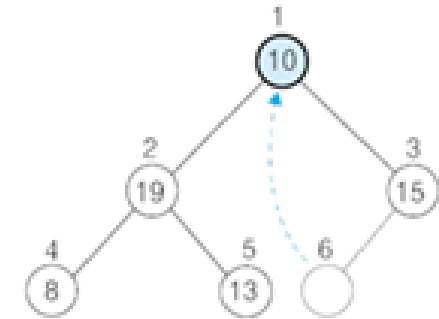
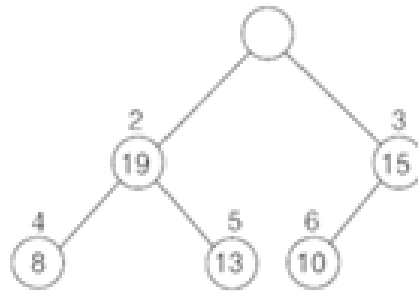
힙에서는 어떤 데이터가
삭제되어야 할까? -> 루트

삭제 후 어떤 조치가 필요한가?
-> 힙으로 구성함이 필요.
완전 이진 트리 && 힙 구조유지

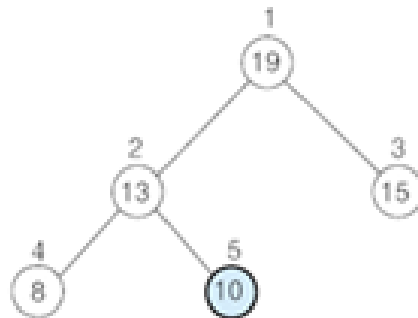
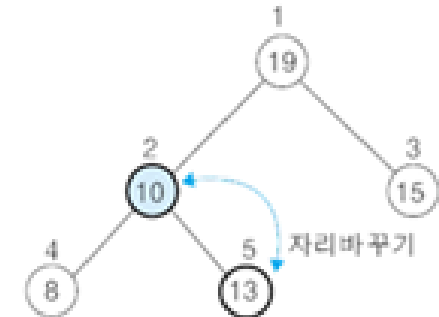
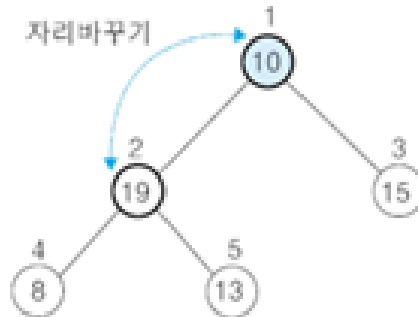
힙으로 구성하기 위해 할 일은?
-> 끝의 노드를 루트로 가져오고,
힙의 구조가 되도록 조정한다.
(루트부터 하향 조정함)

```
While ( 자식과 힙이 아닌 동안 ) {
    두 자식 중 더 큰 놈과 교환
}
```

루트의 원소 삭제



자리바꾸기



Q. 공백 상태에서부터
임의 자료의 삽입 삭제가
수행되는 힙을 운영하시오.

- 힙의 운영 : 힙은 완전 이진트리이므로 배열을 이용하여 운용하면 매우 효율적임

