

Professional 사전 과정

**트리**

# 차례

## 트리

- ✓ 개념
- ✓ 이진트리
- ✓ 트리순회
- ✓ 표현방법
- ✓ 수식트리
- ✓ 이진탐색트리

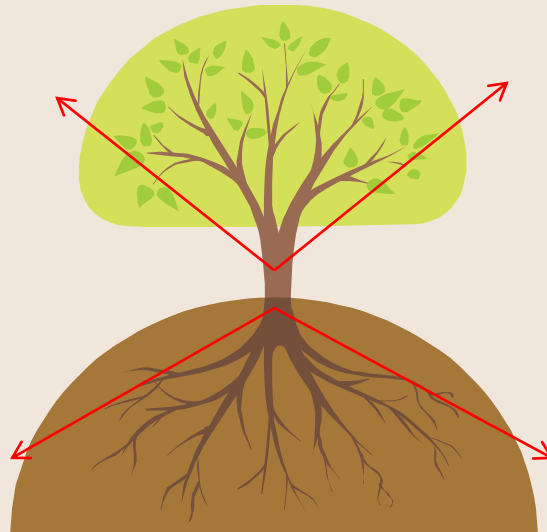
## 힙

**개념**

# 트리

## 트리의 개념


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

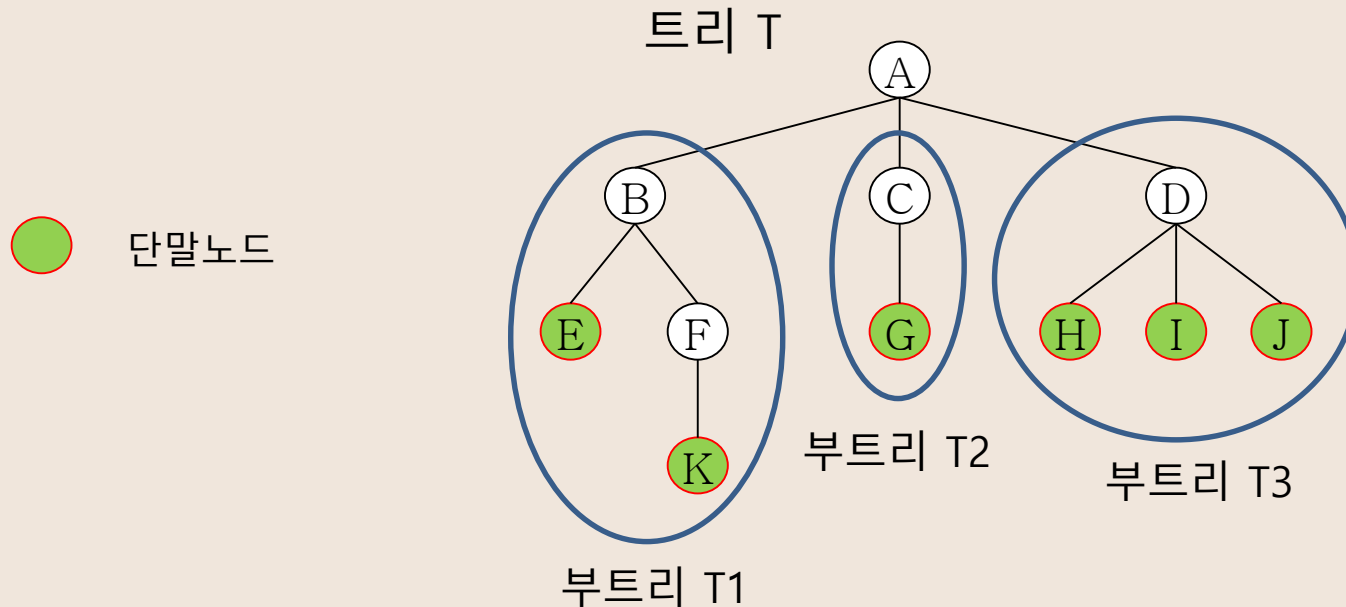


# 트리 - 정의

 한 개 이상의 노드로 이루어진 유한 집합이며 다음 조건을 만족한다.

1. 노드 중 최상위 노드를 루트(root)라 한다.
2. 나머지 노드들은  $n(>= 0)$ 개의 분리 집합  $T_1, \dots, T_N$ 으로 분리될 수 있다.

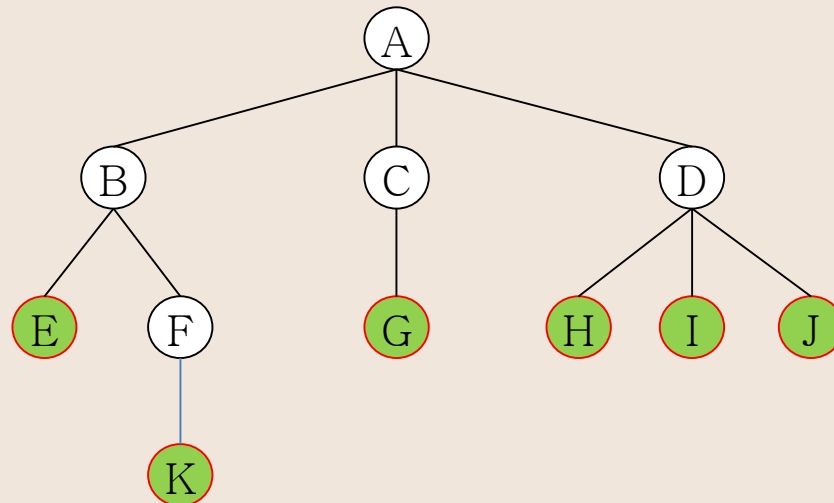
 이들  $T_1, \dots, T_N$ 은 각각 하나의 트리가 되며(재귀적 정의) 루트의 부트리(subtree)라 한다.



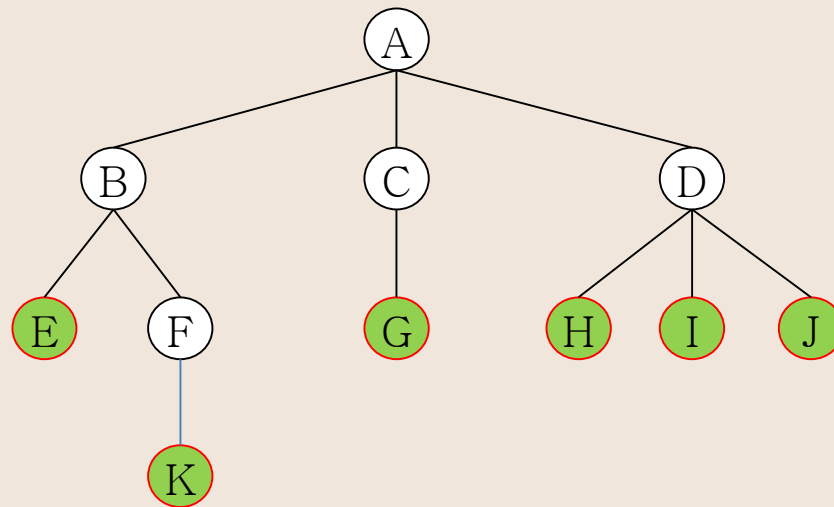
# 트리-용어정리

- ✓ 노드(node) – 트리의 원소
  - ✦ 트리 T의 노드 - A,B,C,D,E,F,G,H,I,J,K
- ✓ 간선(edge) – 노드를 연결하는 선. 부모 노드와 자식 노드를 연결
- ✓ 루트 노드(root node) – 트리의 시작 노드
  - ✦ 트리 T의 루트노드 - A

트리 T



- ✓ 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
- ✓ 자손 노드 – 서브 트리에 있는 하위 레벨의 노드들
  - ✦ B의 자손 노드 – E,F,K



## ✓ 차수(degree)

✦ 노드의 차수 : 노드에 연결된 자식 노드의 수.

✍ B의 차수=2, C의 차수=1

✦ 트리의 차수 : 트리에 있는 노드의 차수 중에서 가장 큰 값

✍ 트리 T의 차수=3

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

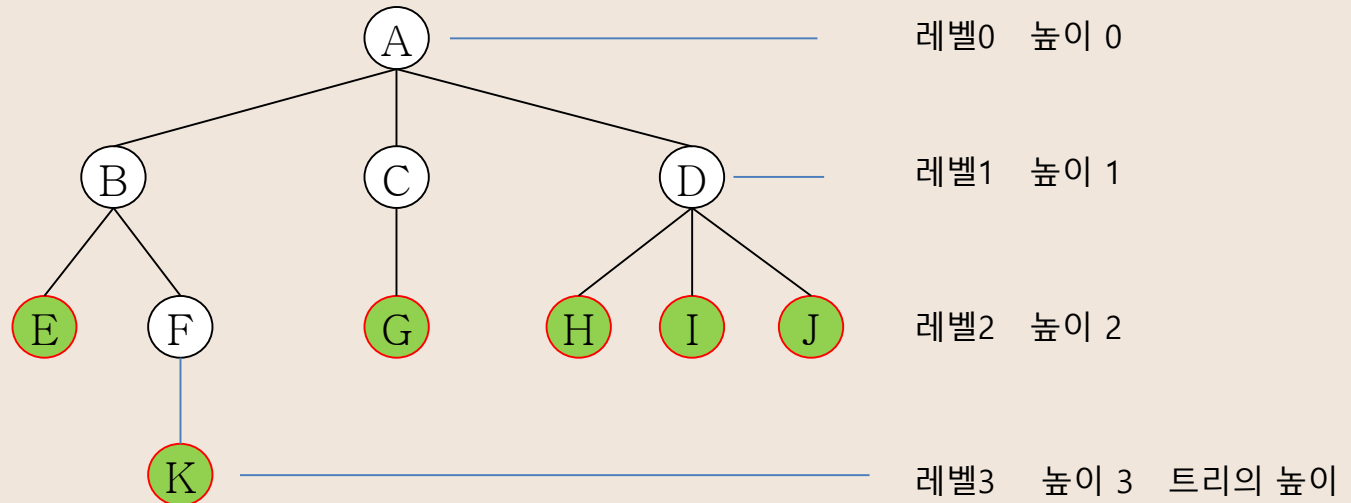
## ✓ 높이

✦ 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 T의 높이=3

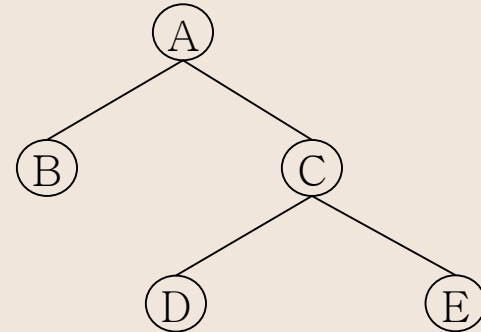
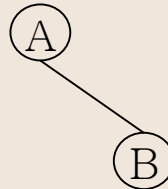
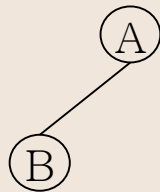






**이진트리**

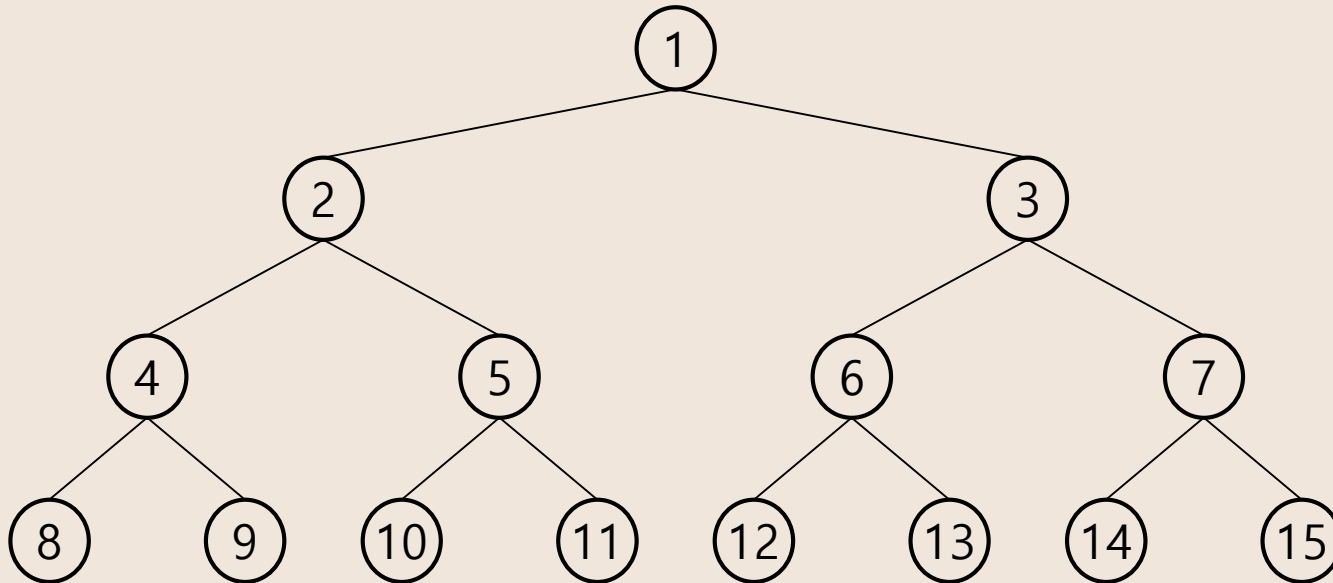
# 이진트리

- ✎ 모든 노드들이 2개의 서브트리를 갖는 특별한 형태의 트리
- ✎ 각 노드가 자식 노드를 최대한 2개 까지만 가질 수 있는 트리
  - ✓ 왼쪽 자식 노드(left child node)
  - ✓ 오른쪽 자식 노드(right child node)
- ✎ 이진 트리의 예



# 이진트리 - 특성

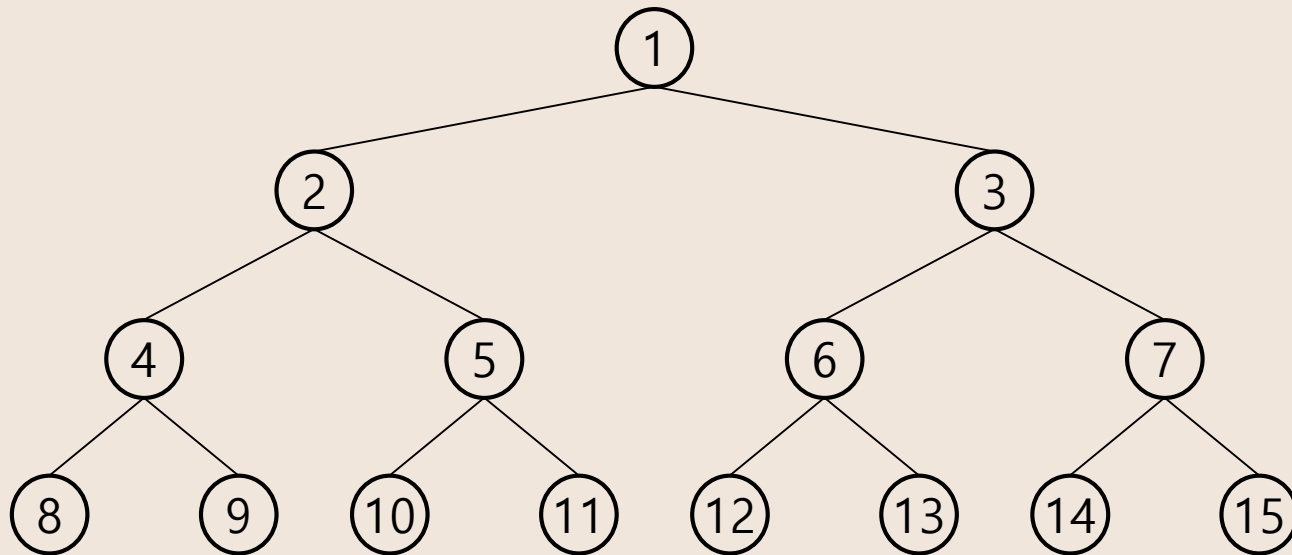
-  레벨  $i$ 에서의 노드의 최대 개수는  $2^i$ 개
-  높이가  $h$ 인 이진 트리가 가질 수 있는 노드의 최소 개수는  $(h+1)$ 개가 되며, 최대 개수는  $(2^{h+1}-1)$ 개가 된다.



# 이진트리 - 종류

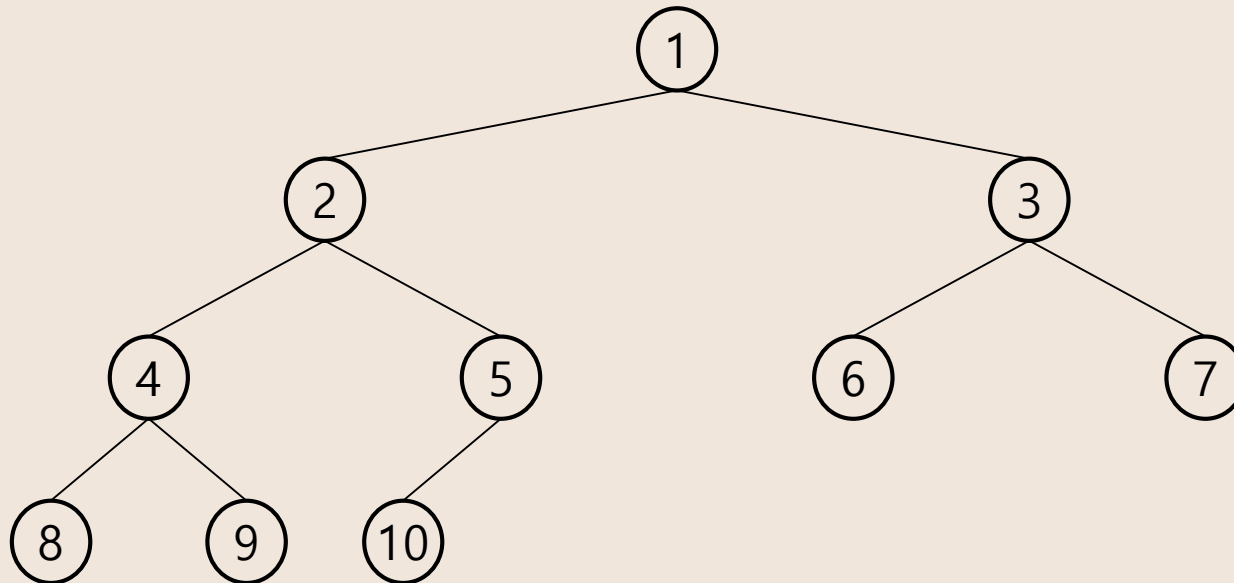
## 포화 이진 트리(Full Binary Tree)

- ✓ 모든 레벨에 노드가 포화상태로 차 있는 이진 트리
- ✓ 높이가  $h$ 일 때, 최대의 노드 개수인  $(2^{h+1}-1)$  의 노드를 가진 이진 트리
  - ✧ 높이 3일 때  $2^{3+1}-1 = 15$ 개의 노드
- ✓ 루트를 1번으로 하여  $2^{h+1}-1$ 까지 정해진 위치에 대한 노드 번호를 가짐



## ✎ 완전 이진 트리(Complete Binary Tree)

- ✓ 높이가  $h$ 이고 노드 수가  $n$ 개일 때 (단,  $2^h \leq n < 2^{h+1}-1$ ), 포화 이진 트리의 노드 번호 1번부터  $n$ 번까지 빈 자리가 없는 이진 트리
- ✓ 예) 노드가 10개인 완전 이진 트리

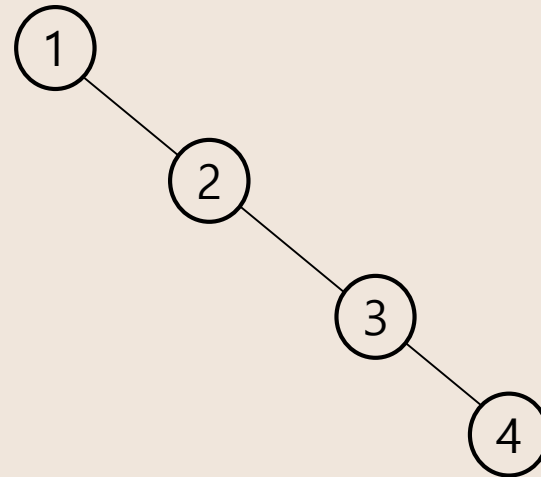
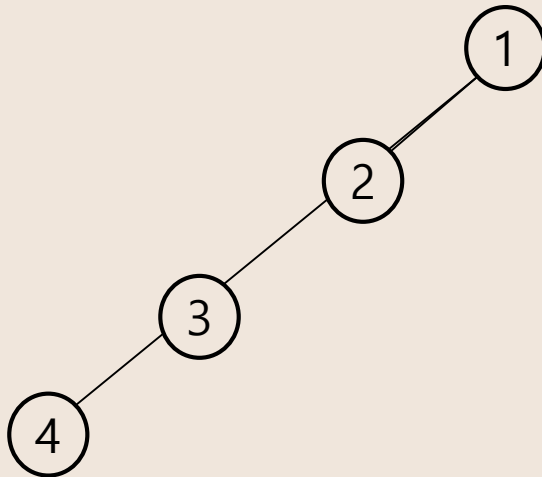


## 편향 이진 트리(Skewed Binary Tree)

- ✓ 높이  $h$ 에 대한 최소 개수의 노드를 가지면서 한쪽 방향의 자식 노드 만을 가진 이진 트리

✦ 왼쪽 편향 이진 트리

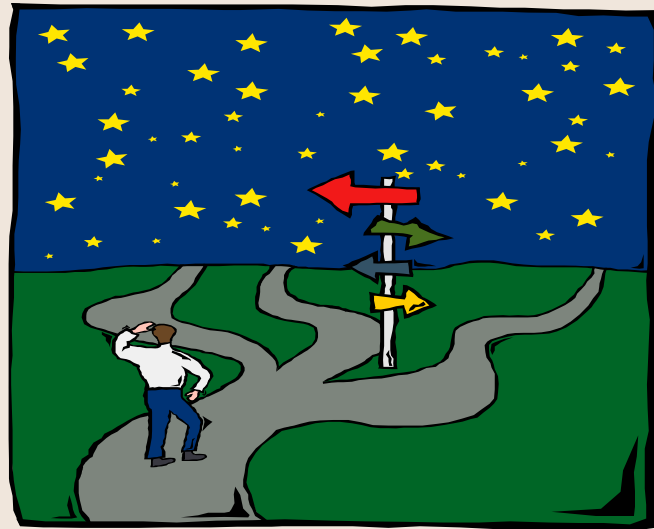
✦ 오른쪽 편향 이진 트리



**트리순회**

# 이진트리 - 순회 (traversal)

- ✎ 순회(traversal)란 트리의 각 노드를 중복되지 않게 전부 방문(visit)하는 것을 말하는데 트리는 비 선형 구조이기 때문에 선형구조에서와 같이 선후 연결 관계를 알 수 없다.
- ✎ 따라서 특별한 방법이 필요하다.





✎ 순회(traversal): 트리의 노드들을 체계적으로 방문하는 것

✎ 3가지의 기본적인 순회방법

✓ 전위순회(preorder traversal) : VLR

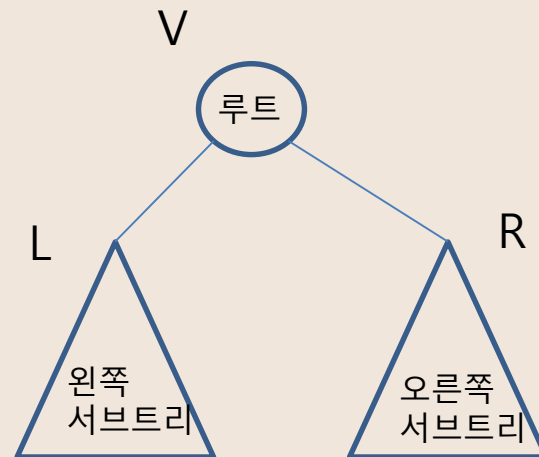
✦ 자손노드보다 루트노드를 먼저 방문한다.

✓ 중위순회(inorder traversal) : LVR

✦ 왼쪽 자손, 루트, 오른쪽 자손 순으로 방문한다.

✓ 후위순회(postorder traversal) : LRV

✦ 루트노드보다 자손을 먼저 방문한다.



## 전위 순회(preorder traversal)

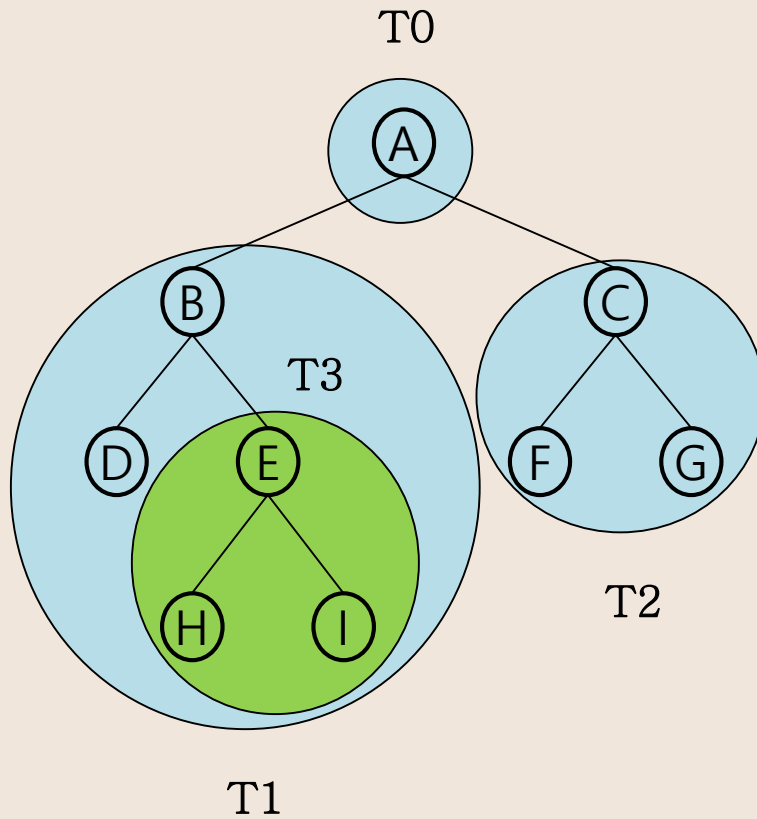
### ✓ 수행 방법

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

### ✓ 전위 순회 알고리즘

```
preorder_traverse (T)
  if (T is not null)
  {
    visit(T);
    preorder_traverse(T.left);
    preorder_traverse(T.right);
  }
End preorder_traverse
```

## ✍ 전위순회 예



순서1 :  $T0 \rightarrow T1 \rightarrow T2$

순서2:  $A \rightarrow B \ D \ (T3) \rightarrow C \ F \ G$

총순서: A B D E H I C F G

## 중위 순회(inorder traversal)

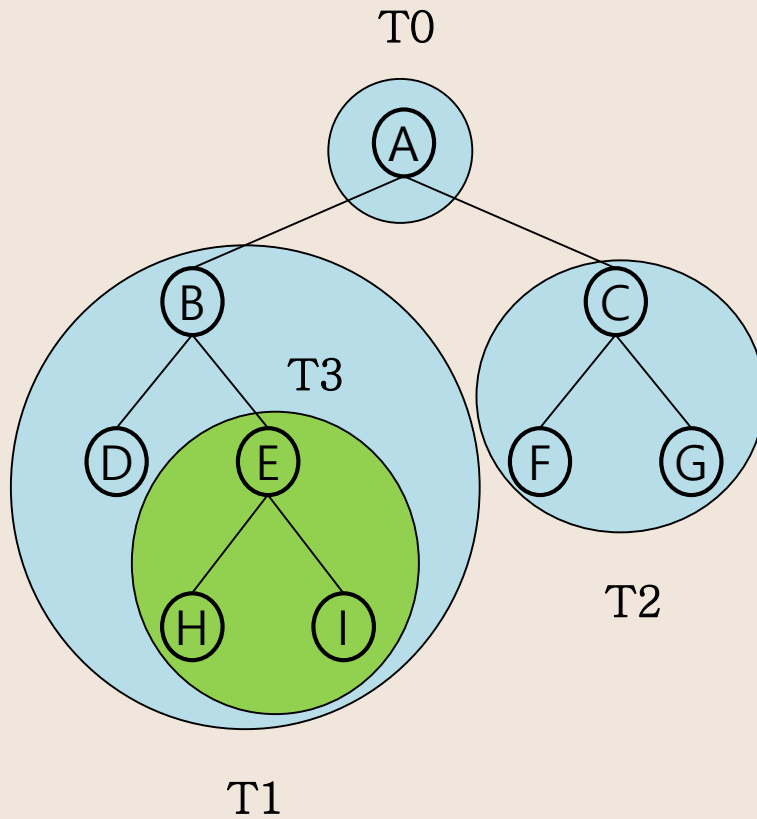
### ✓ 수행 방법

- ① 현재 노드 n의 왼쪽 서브트리로 이동한다. : L
- ② 현재 노드 n을 방문하여 처리한다. : V
- ③ 현재 노드 n의 오른쪽 서브트리로 이동한다. : R

### ✓ 중위 순회 알고리즘

```
inorder_traverse (T)
  if (T is not null)
  {
    inorder_traverse(T.left);
    visit(T);
    inorder_traverse(T.right);
  }
End inorder_traverse
```

## 중위 순회의 예



순서1 : T1 -> T0 -> T2

순서2: D B (T3) -> A -> F C G

총순서: D B H E I A F C G

## 후위 순회(postorder traversal)

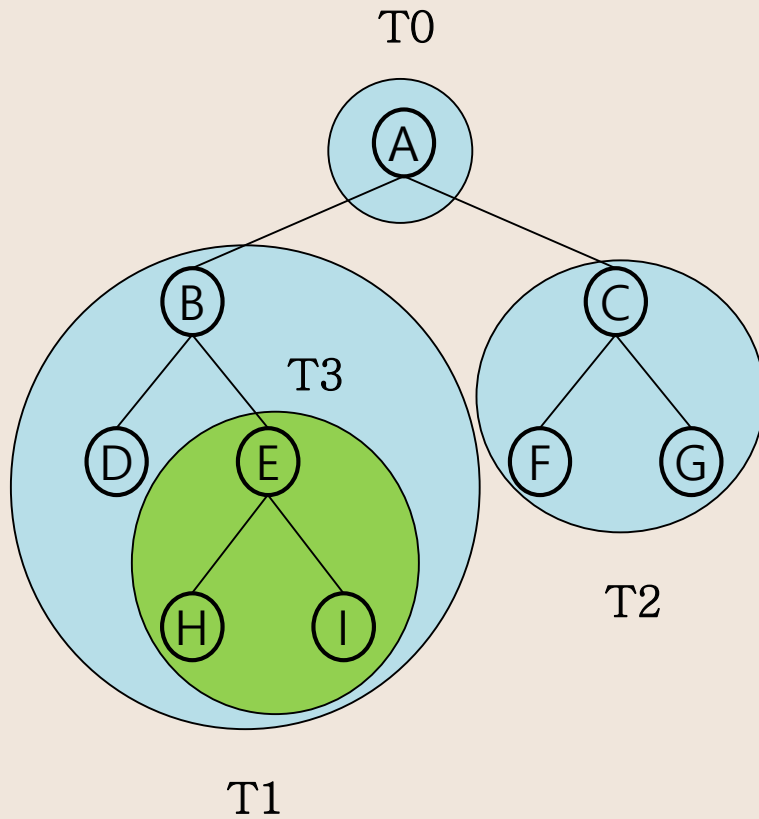
### ✓ 수행 방법

- ① 현재 노드 n의 왼쪽 서브트리로 이동한다. : L
- ② 현재 노드 n의 오른쪽 서브트리로 이동한다. : R
- ③ 현재 노드 n을 방문하여 처리한다. : V

### ✓ 후위 순회 알고리즘

```
postorder_traverse (T)
  if (T is not null)
  {
    postorder_traverse(T.left);
    postorder_traverse(T.right);
    visit(T);
  }
End postorder_traverse
```

## ✍ 후위 순회의 예



순서1 : T1 -> T2 -> T0

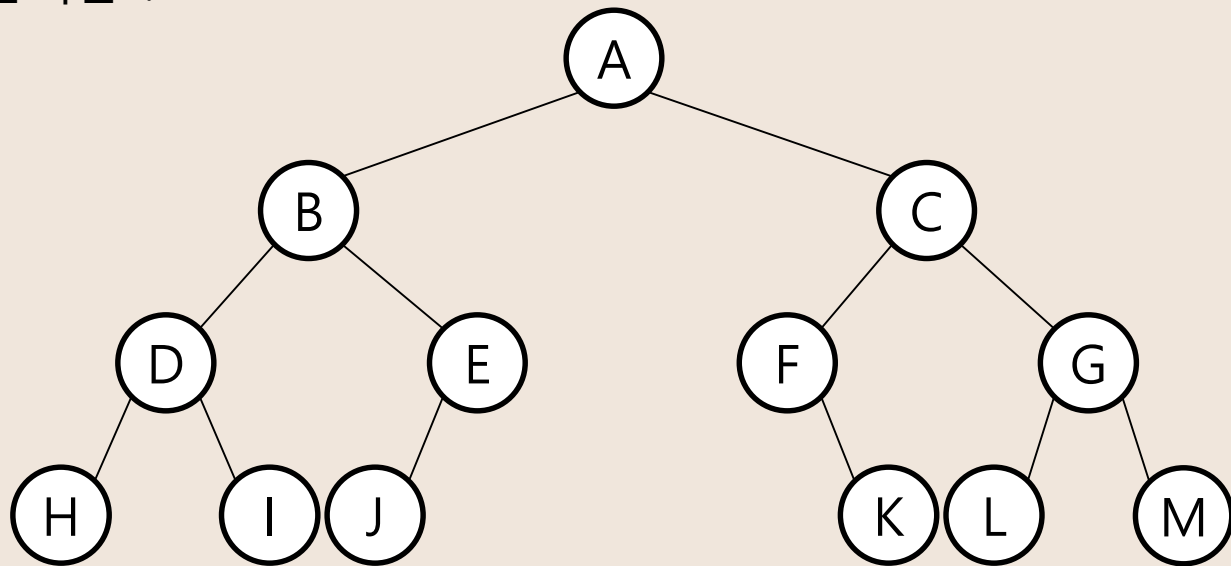
순서2: D (T3) B -> F G C -> A

총순서: D H I E B F G C A

# 이진트리 - 순회 연습 문제

## 이진 트리의 순회

- ✓ 전위 순회는 ?
- ✓ 중위 순회는 ?
- ✓ 후위 순회는 ?



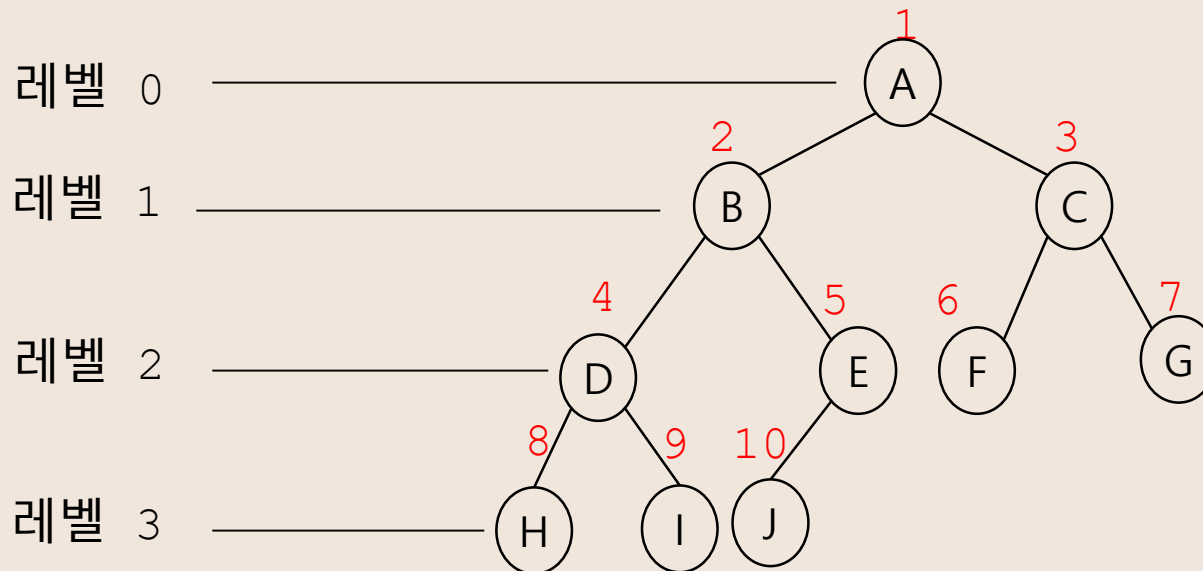


**표현방법**

# 트리의 표현

## 배열을 이용한 이진 트리의 표현

- ✓ 이진 트리에 각 노드 번호를 다음과 같이 부여
- ✓ 루트의 번호를 1로함
- ✓ 레벨  $n$ 에 있는 노드에 대하여 왼쪽부터 오른쪽으로  $2^n$  부터  $2^{n+1} - 1$  까지 번호를 차례로 부여

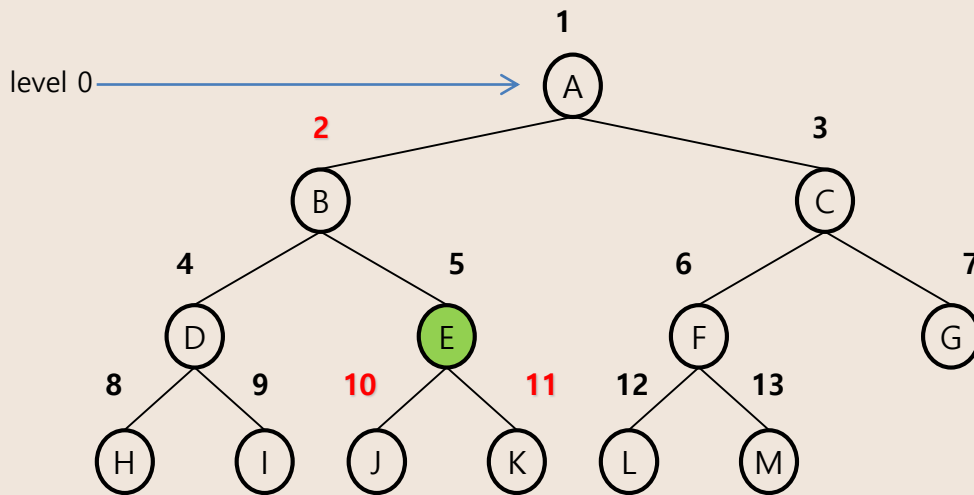


# 트리의 표현 - 배열

## 배열을 이용한 이진 트리의 표현

### 노드 번호의 성질

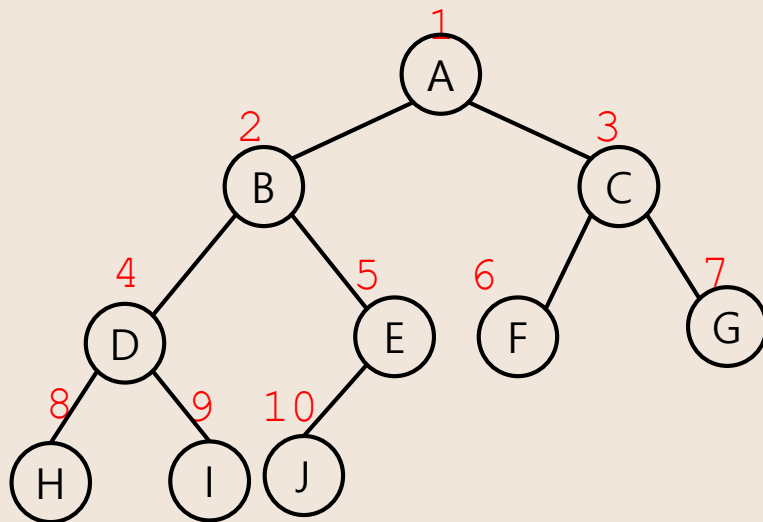
- ✓ 노드 번호가  $i$  인 노드의 부모 노드 번호?  $\lfloor i/2 \rfloor$
- ✓ 노드 번호가  $i$  인 노드의 왼쪽 자식 노드 번호?  $2*i$
- ✓ 노드 번호가  $i$  인 노드의 오른쪽 자식 노드 번호?  $2*i+1$
- ✓ 레벨  $n$ 의 노드 번호 시작 번호는?  $2^n$



0		
1	A	
2	B	부모노드의 인덱스 = 2
3	C	
4	D	
5	E	
6	F	
7	G	
8	H	
9	I	
10	J	왼쪽 자식노드의 인덱스 = 10
11	K	오른쪽 자식노드의 인덱스 = 11
12	L	
13	M	

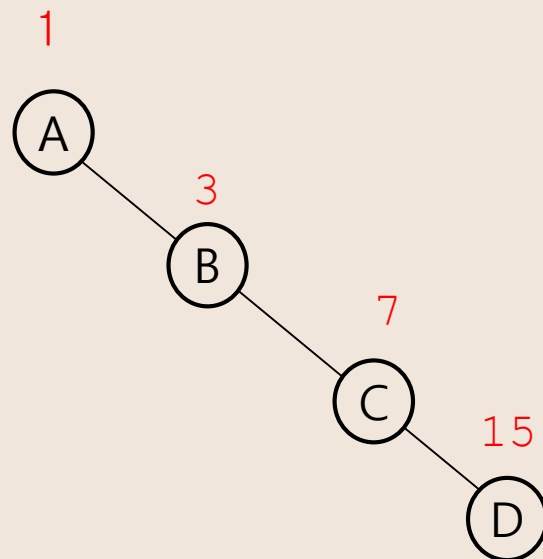
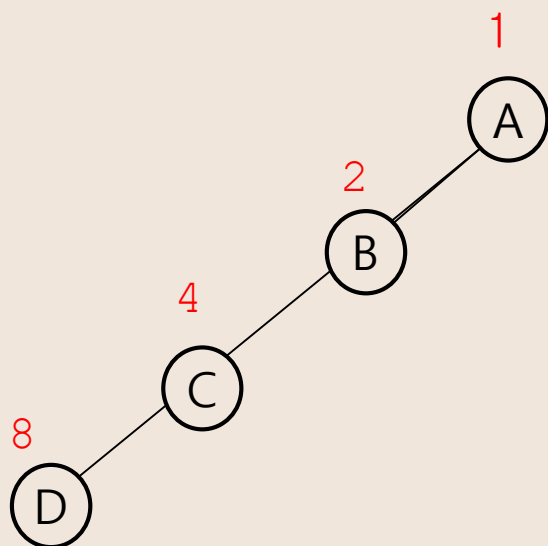
## ✍ 배열을 이용한 이진 트리의 표현

- ✓ 노드 번호를 배열의 인덱스로 사용
- ✓ 높이가  $h$  인 이진 트리를 위한 배열의 크기는?
  - ✦ 레벨  $i$ 의 최대 노드 수는?  $2^i$
  - ✦ 따라서  $1 + 2 + 4 + 8 \dots + 2^i = \sum 2^i = 2^{h+1}-1$



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	A	B	C	D	E	F	G	H	I	J	-	-	-	-	-

## ✍ 배열을 이용한 이진 트리의 표현




0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	A	B	-	C	-	-	-	D	-	-	-	-	-	-	-

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	A	-	B	-	-	-	C	-	-	-	-	-	-	-	D

## 배열을 이용한 이진 트리의 표현의 단점

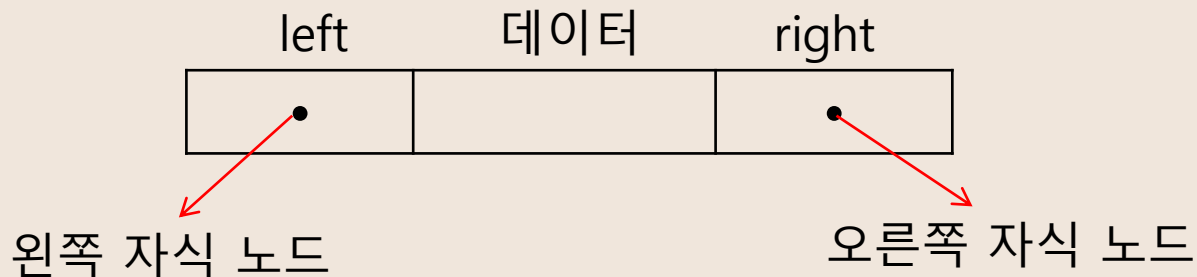
- ✓ 편향 이진 트리의 경우에 사용하지 않는 배열 원소에 대한 메모리 공간 낭비 발생
- ✓ 트리의 중간에 새로운 노드를 삽입하거나 기존의 노드를 삭제할 경우 배열의 크기 변경 어려워 비효율적

# 트리의 표현 - 연결리스트

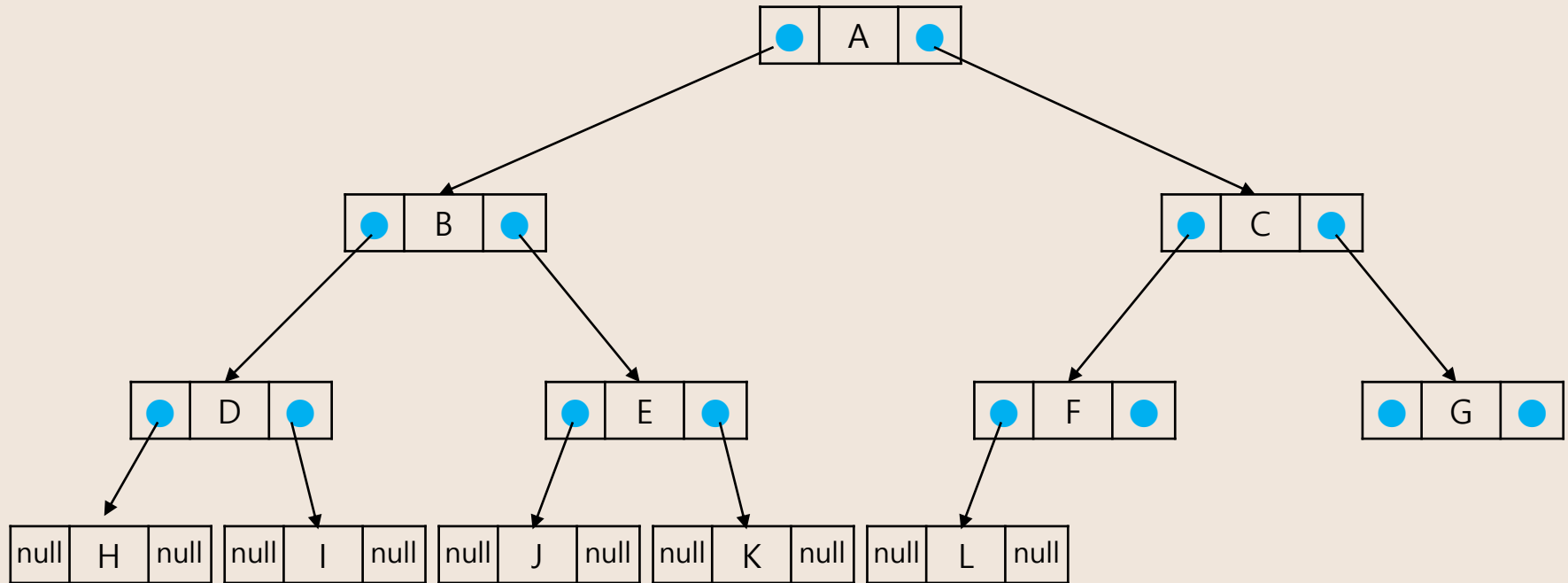
 배열을 이용한 이진 트리의 표현의 단점을 보완하기 위해 연결리스트를 이용하여 트리를 표현할 수 있다.

 연결 자료구조를 이용한 이진트리의 표현

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





## ✎ 완전 이진 트리의 연결 리스트 표현





## 연습문제

-  첫줄에는 트리의 정점의 총 수  $V$ 가 주어진다. 그 다음 줄에는  $V-1$ 개 간선이 나열된다. 간선은 그것을 이루는 두 정점으로 표기된다. 간선은 항상 “부모 자식” 순서로 표기된다. 아래 예에서 두 번째 줄 처음 1 2는 정점 1과 2를 잇는 간선을 의미하며 1이 부모, 2가 자식을 의미한다. 간선은 부모 정점 번호가 작은 것부터 나열되고, 부모 정점이 동일하다면 자식 정점 번호가 작은 것부터 나열된다.
-  다음 이진 트리 표현에 대하여 전위 순회하여 정점의 번호를 출력하시오.

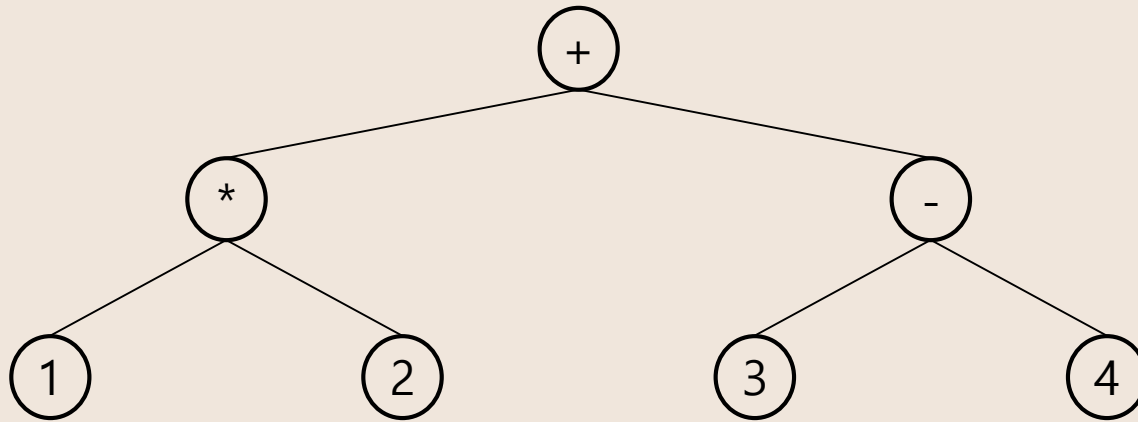
13 ← 정점의 개수

1 2 1 3 2 4 3 5 3 6 4 7 5 8 5 9 6 10 6 11 7 12 11 13

# 수식트리

# 수식트리

- ✎ 수식을 표현하는 이진 트리
- ✎ 수식 이진 트리(Expression Binary Tree)라고 부르기도 함.
- ✎ 연산자는 루트 노드이거나 가지 노드
- ✎ 피연산자는 모두 잎 노드

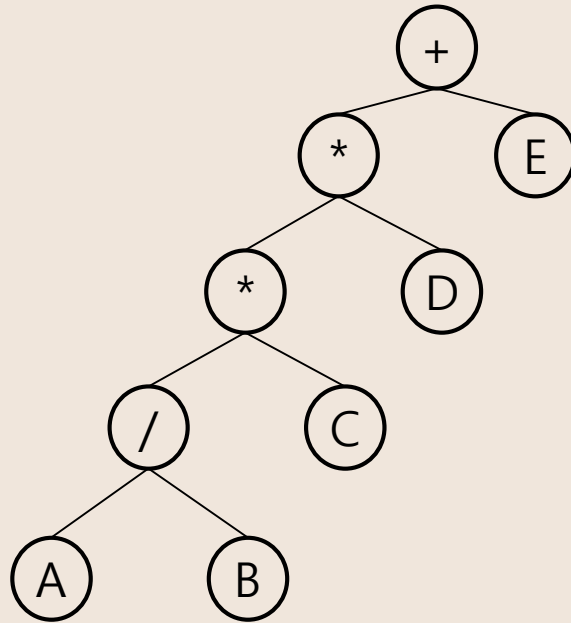


# 수식트리의 순회

 중위 순회:  $A / B * C * D + E$  [식의 중위 표기법]

 후위 순회:  $A B / C * D * E +$  [식의 후위 표기법]

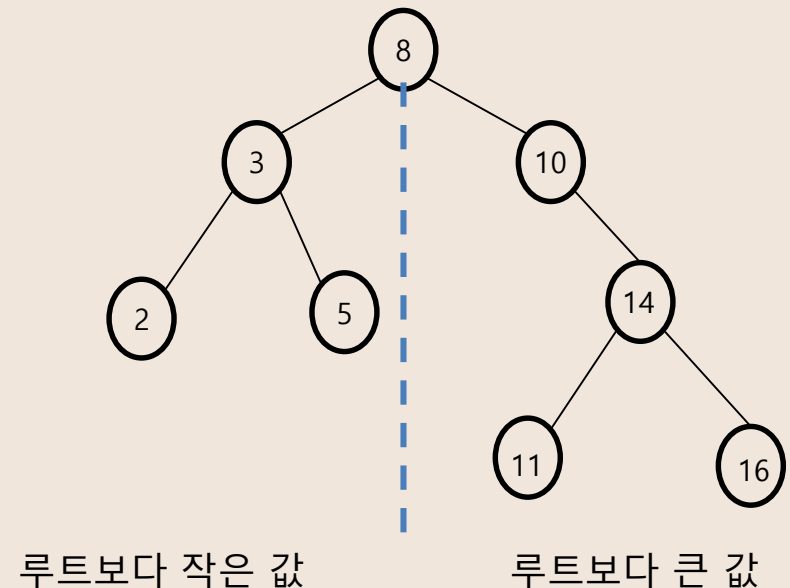
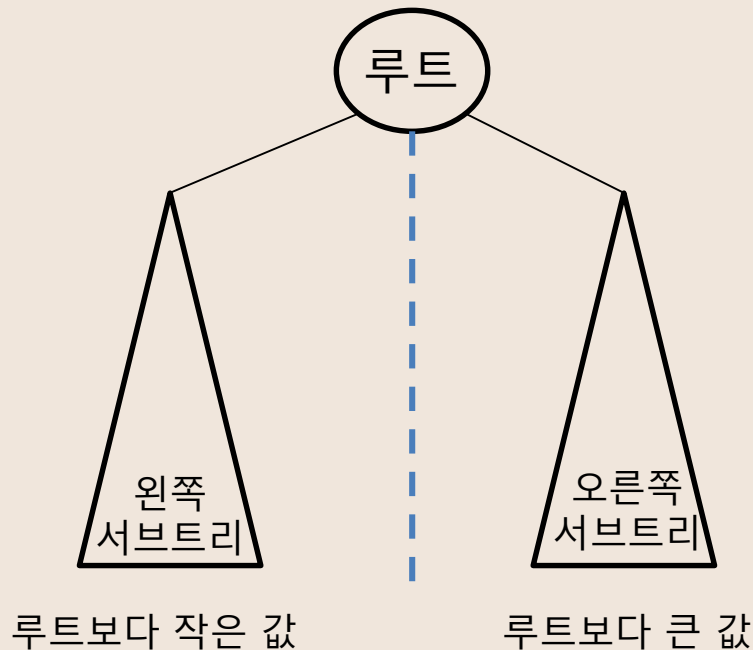
 전위 순회:  $+ * * / A B C D E$  [식의 전위 표기법]



**이진탐색트리**

# 이진탐색트리

- ✎ 탐색작업을 효율적으로 하기 위한 자료구조
- ✎ 모든 원소는 서로 다른 유일한 키를 갖는다.
- ✎  $\text{key}(\text{왼쪽 서브트리}) < \text{key}(\text{루트 노드}) < \text{key}(\text{오른쪽 서브트리})$
- ✎ 왼쪽 서브트리와 오른쪽 서브트리도 이진 탐색 트리다.
- ✎ 중위 순회하면 오름차순으로 정렬된 값을 얻을 수 있다.



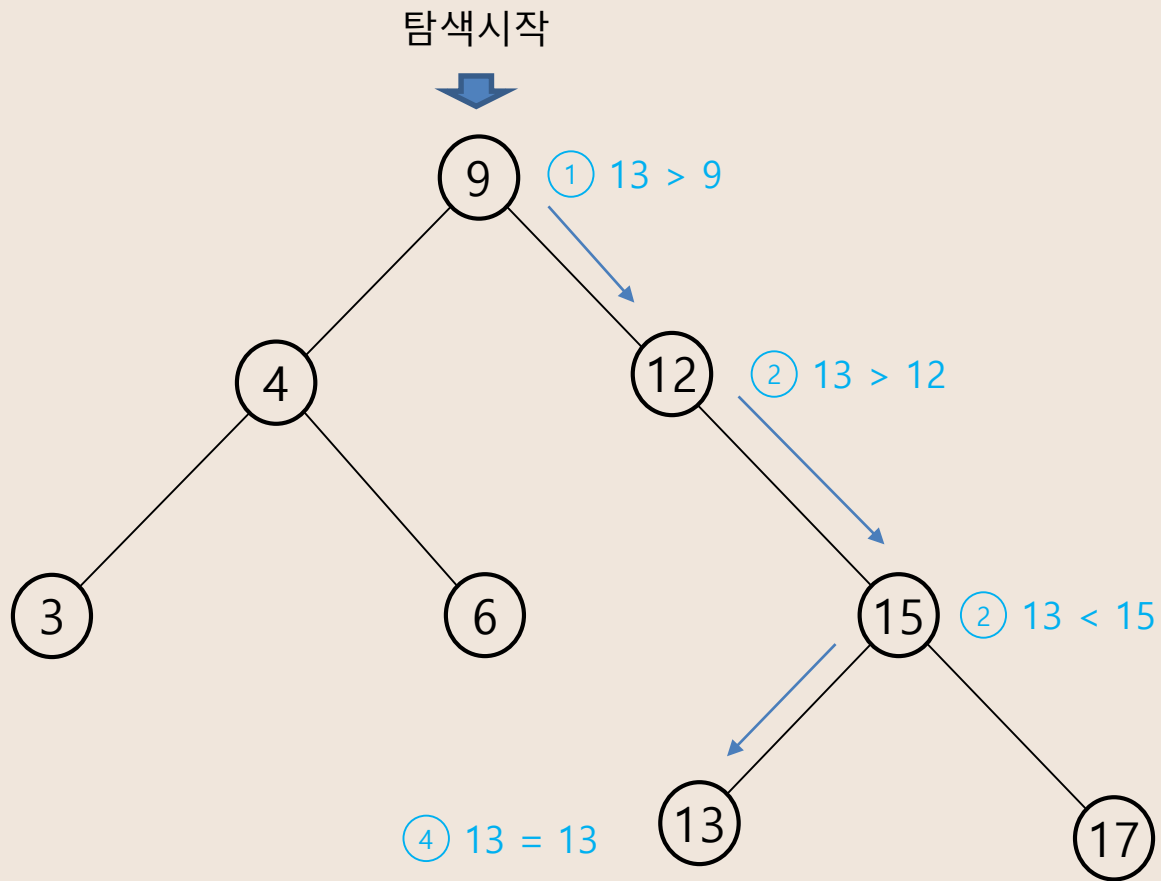
# 이진탐색트리 - 연산

## 탐색연산

- ✓ 루트에서 시작한다.
- ✓ 탐색할 키값  $x$ 를 루트 노드의 키값과 비교한다.
  - ★ (키값  $x =$  루트노드의 키값)인 경우 :
    - ➔ 원하는 원소를 찾았으므로 탐색연산 성공
  - ★ (키값  $x <$  루트노드의 키값)인 경우 :
    - ➔ 루트노드의 왼쪽 서브트리에 대해서 탐색연산 수행
  - ★ (키값  $x >$  루트노드의 키값)인 경우 :
    - ➔ 루트노드의 오른쪽 서브트리에 대해서 탐색연산 수행
- ✓ 서브트리에 대해서 순환적으로 탐색 연산을 반복한다.

## 탐색연산

✓ 13 탐색





## 삽입 연산

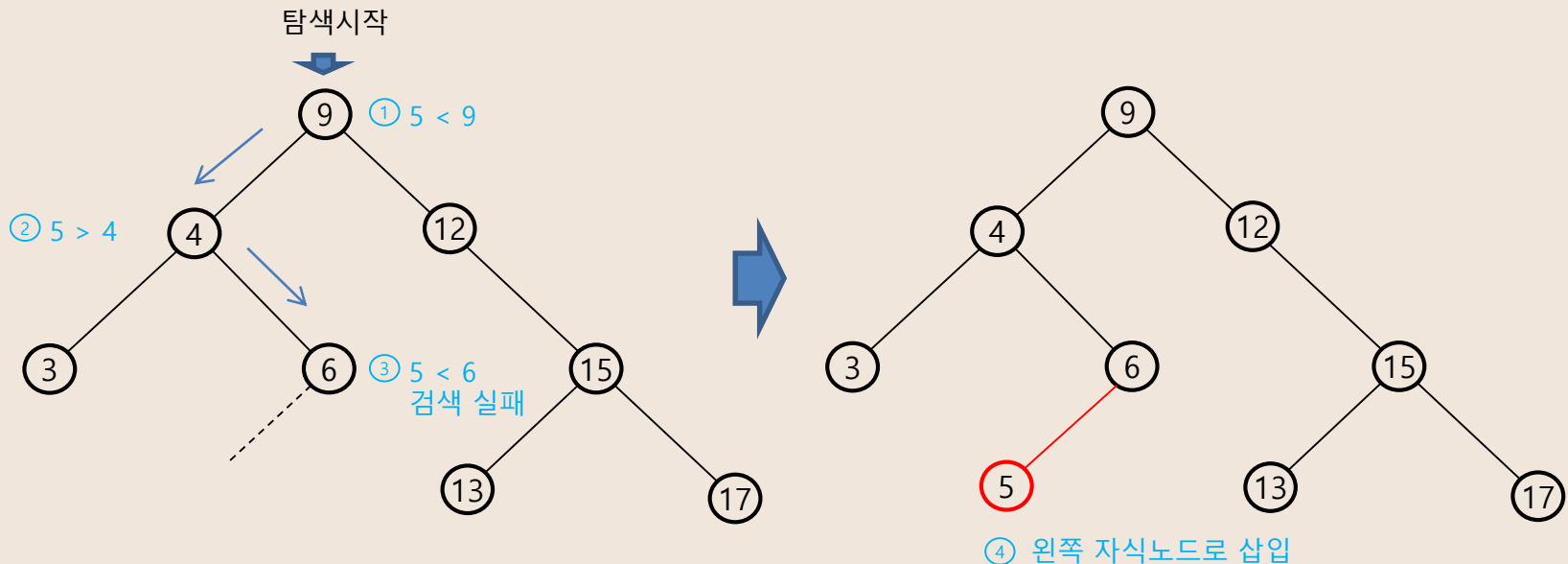
1) 먼저 탐색 연산을 수행

✦ 삽입할 원소와 같은 원소가 트리에 있으면 삽입할 수 없으므로, 같은 원소가 트리에 있는지 탐색하여 확인한다.


✦ 탐색에서 탐색 실패가 결정되는 위치가 삽입 위치가 된다.

2) 탐색 실패한 위치에 원소를 삽입한다.


➤ 다음 예는 5를 삽입하는 예이다



# 이진탐색트리 - 성능


 탐색(searching), 삽입(insertion), 삭제(deletion) 시간은 트리의 높  
이 만큼 시간이 걸린다.

✓  $O(h)$ ,  $h$  : BST의 깊이(height)

 평균의 경우

✓ 이진 트리가 균형적으로 생성되어 있는 경우

✓  $O(\log n)$

 최악의 경우

✓ 한쪽으로 치우친 경사 이진트리의 경우

✓  $O(n)$

✓ 순차탐색과 시간복잡도가 같다.

## 검색알고리즘의 비교

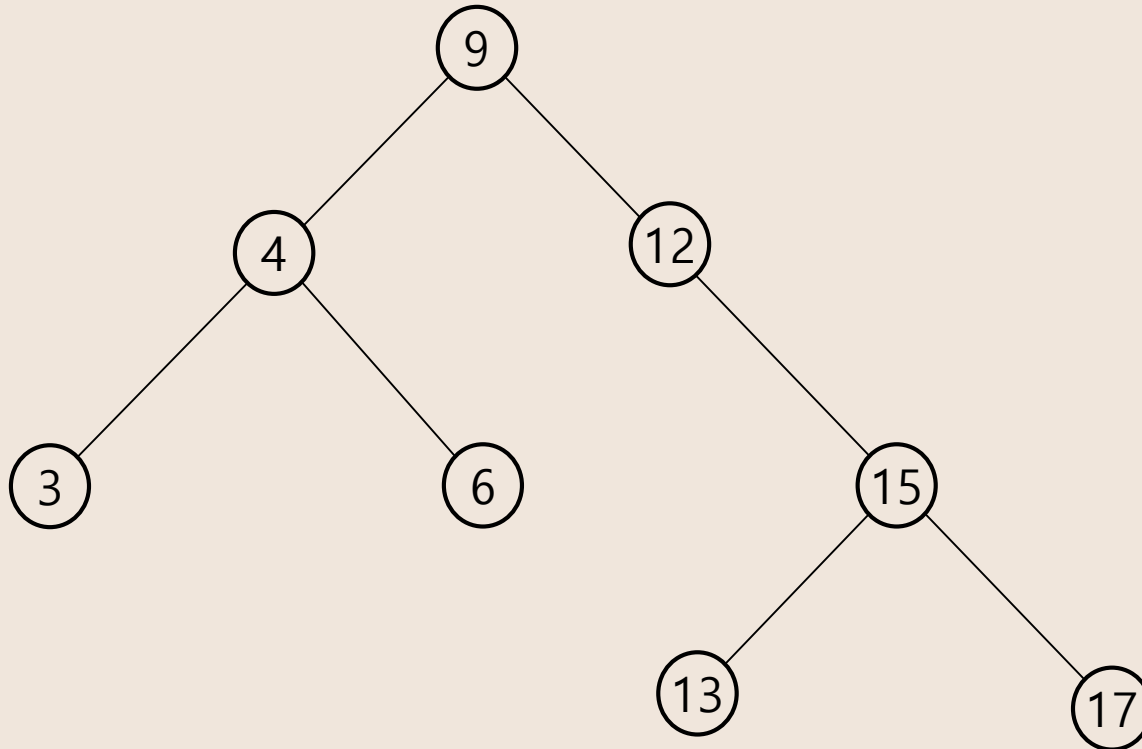
- ✓ 배열에서의 순차 검색 :  $O(N)$
- ✓ 정렬된 배열에서의 순차 검색 :  $O(N)$
- ✓ 정렬된 배열에서의 이진탐색 :  $O(\log N)$ 
  - ✦ 고정 배열 크기와 삽입, 삭제 시 추가 연산 필요
- ✓ 이진 탐색트리에서의 평균 :  $O(\log N)$ 
  - ✦ 최악의 경우 :  $O(N)$
  - ✦ 완전 이진 트리 또는 균형트리로 바꿀 수 있다면 최악의 경우를 없앨 수 있다.
    - ✦ 새로운 원소를 삽입할 때 삽입 시간을 줄인다.
    - ✦ 평균과 최악의 시간이 같다.  $O(\log n)$
- ✓ 해쉬 검색 :  $O(1)$ 
  - ✦ 추가 저장 공간이 필요

## 상용에서 검색을 위해 어떤 알고리즘을 사용할까?

# 이진탐색트리 – 연산 연습


## 삭제 연산

- ✓ 삭제연산에 대해 알고리즘을 생각해 봅시다.
- ✓ 다음 트리에 대하여 13, 12, 9를 차례로 삭제해 보자.



**힙 (HEAP)**

## 힙(heap)

 완전 이진 트리에 있는 노드 중에서 키값이 가장 큰 노드나 키값이 가장 작은 노드를 찾기 위해서 만든 자료구조

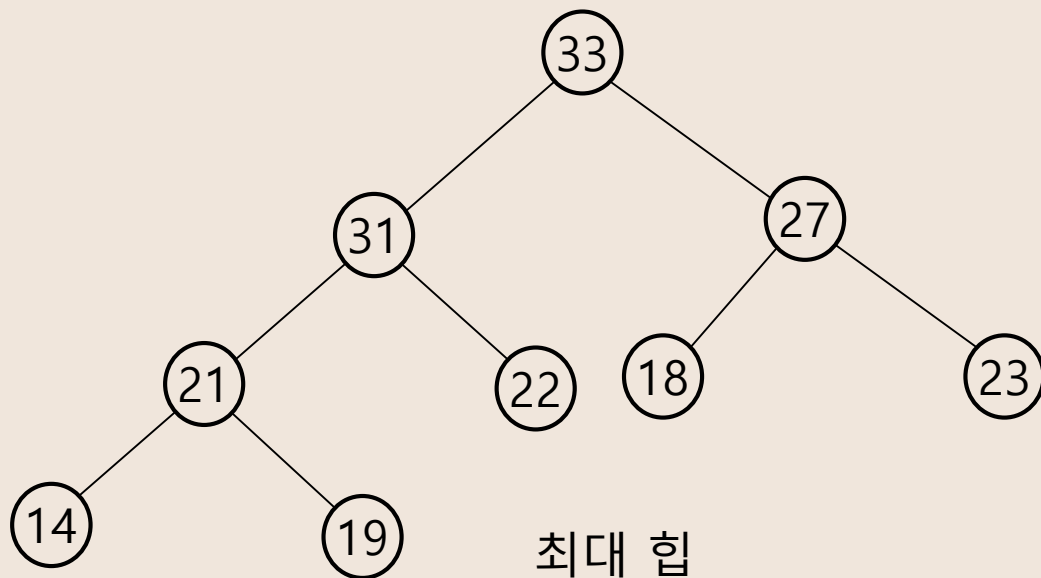
### 최대 힙(max heap)

- ✓ 키값이 가장 큰 노드를 찾기 위한 **완전 이진 트리**
- ✓ {부모노드의 키값 > 자식노드의 키값}
- ✓ 루트 노드 : 키값이 가장 큰 노드

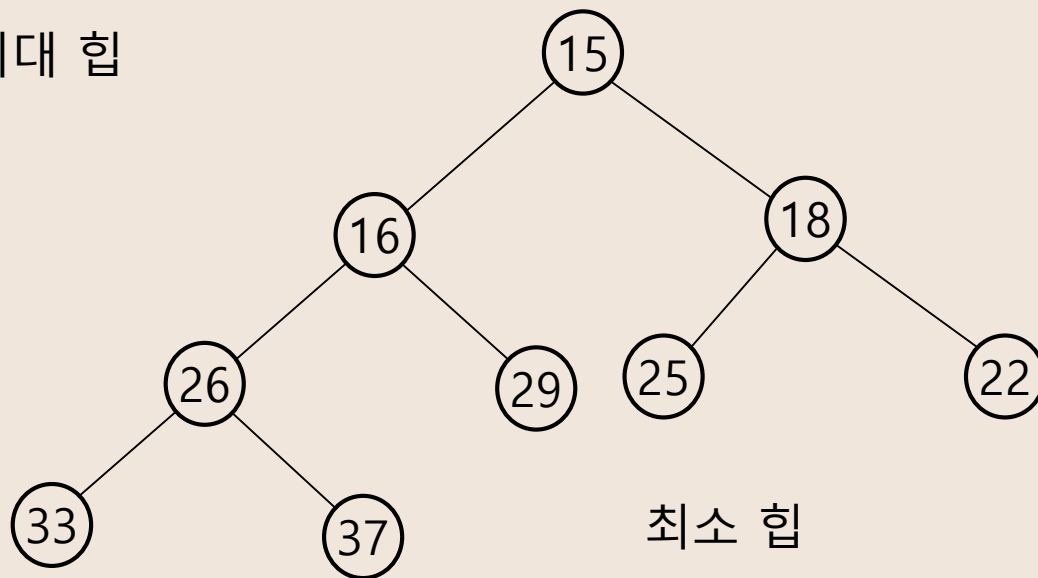
### 최소 힙(min heap)

- ✓ 키값이 가장 작은 노드를 찾기 위한 **완전 이진 트리**
- ✓ {부모노드의 키값 < 자식노드의 키값}
- ✓ 루트 노드 : 키값이 가장 작은 노드

## 힙의 예



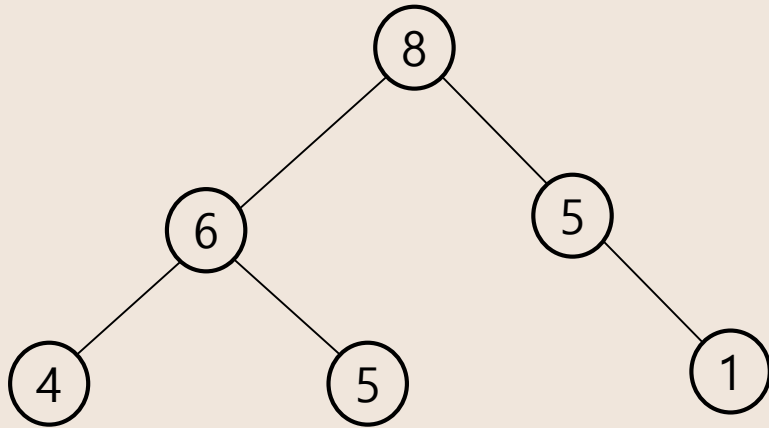
최대 힙



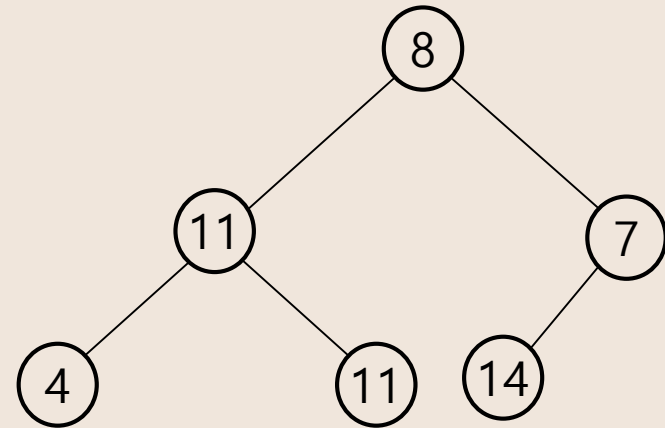
최소 힙

## ✎ 힙이 아닌 이진 트리의 예

✓ 아닌 이유를 설명해 보세요



트리1

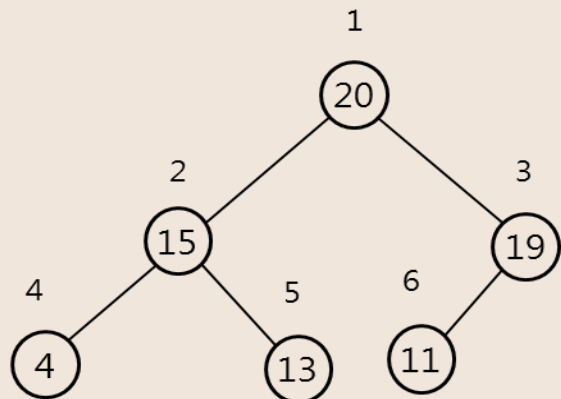


트리2

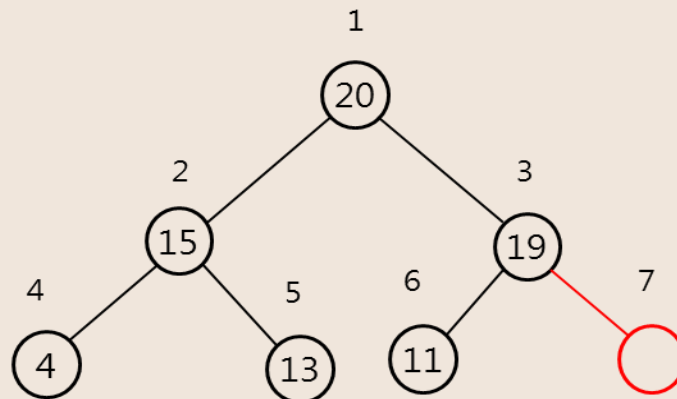


# 💬 힙 연산 - 삽입

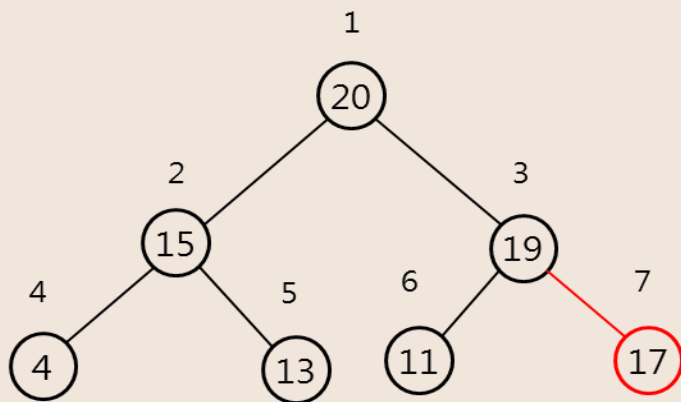
## ✎ 17 삽입



삽입 전의 힙

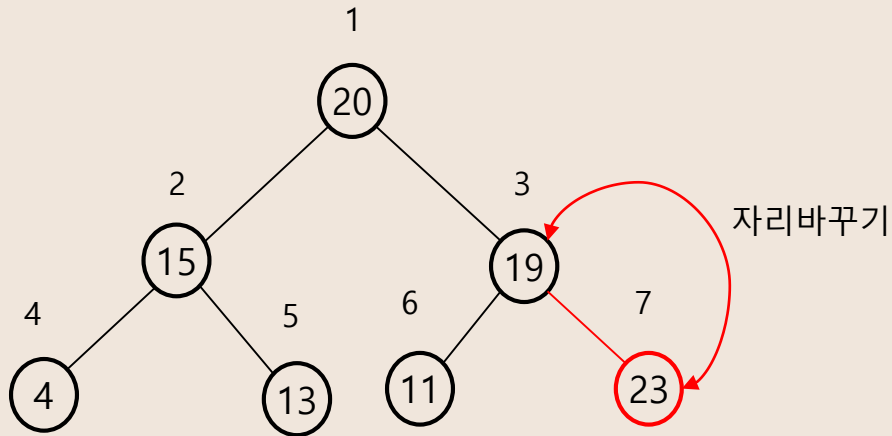


삽입 할 자리 확장

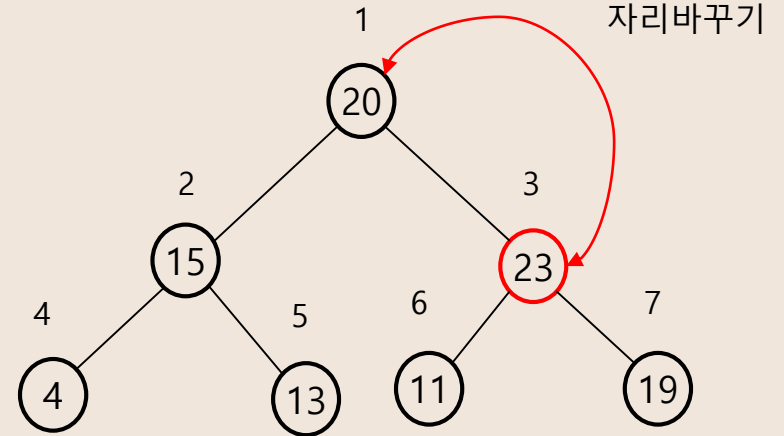


확장한 자리에 삽입할 원소 저장

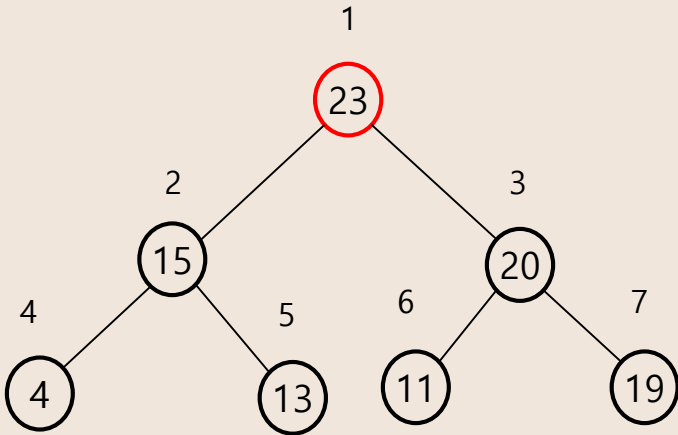
## 23삽입



1) (삽입 노드 23 > 부모 노드 19) : 자리바꾸기



2) (삽입 노드 23 > 부모 노드 19) : 자리바꾸기



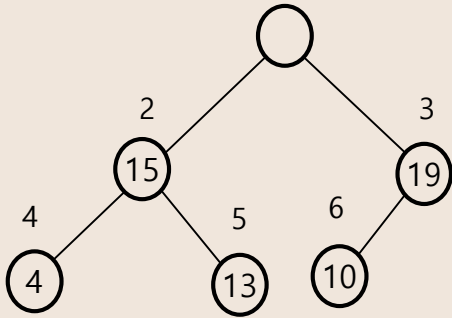
3) 비교할 부모 노드가 없으므로 자리 확정

## 힙 연산 - 삭제

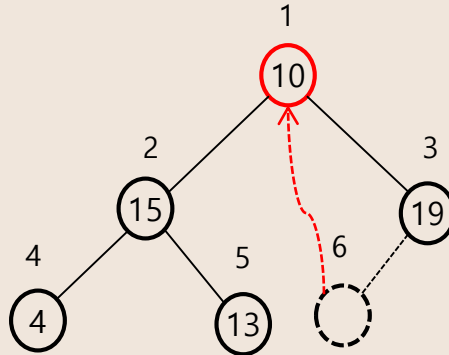
- ✎ 힙에서는 루트 노드의 원소만을 삭제 할 수 있다.
- ✎ 루트 노드의 원소를 삭제하여 반환한다.
- ✎ 힙의 종류에 따라 최대값 또는 최소값을 구할 수 있다.

## ✎ 힙에서의 삭제 예

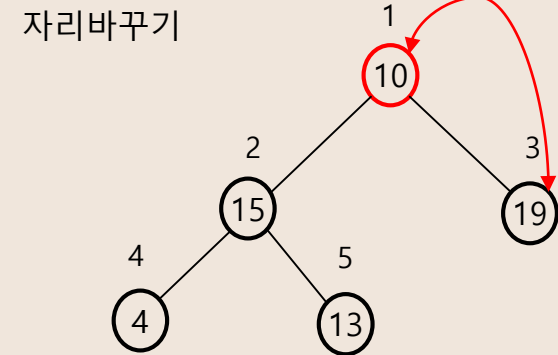
루트의 원소 삭제



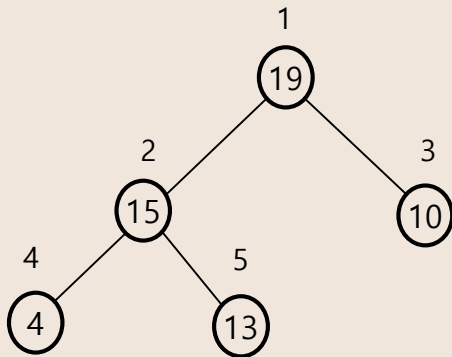
1) 루트 노드의 원소 삭제



2) 마지막 노드 삭제



3) (삽입노드 10 < 자식노드 19) :자리바꾸기

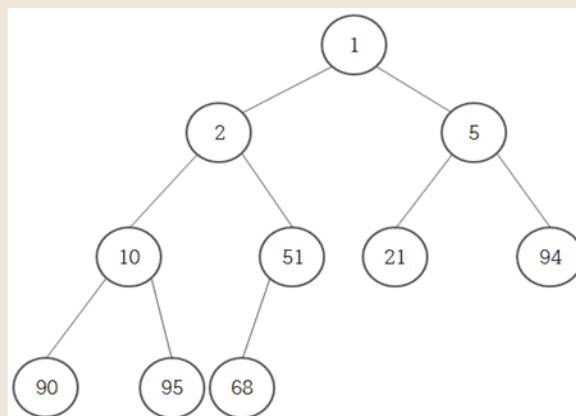


4) 자리 확정

# 💬 힙을 이용한 우선순위 큐

## ✍ 힙(Heap)

- ✓ 완전 이진 트리로 구현된 자료구조로서, 키값이 가장 큰 노드나 가장 작은 노드를 찾기에 유용한 자료구조
- ✓ 아래의 예는 최소 힙(Min heap)으로서, 가장 작은 키값을 가진 노드가 항상 루트에 위치한다.



<최소 힙의 예>

- ✓ 힙의 키를 우선순위로 활용하여 우선순위 큐를 구현할 수 있다.
- ✓ 관련 링크:

<http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>