

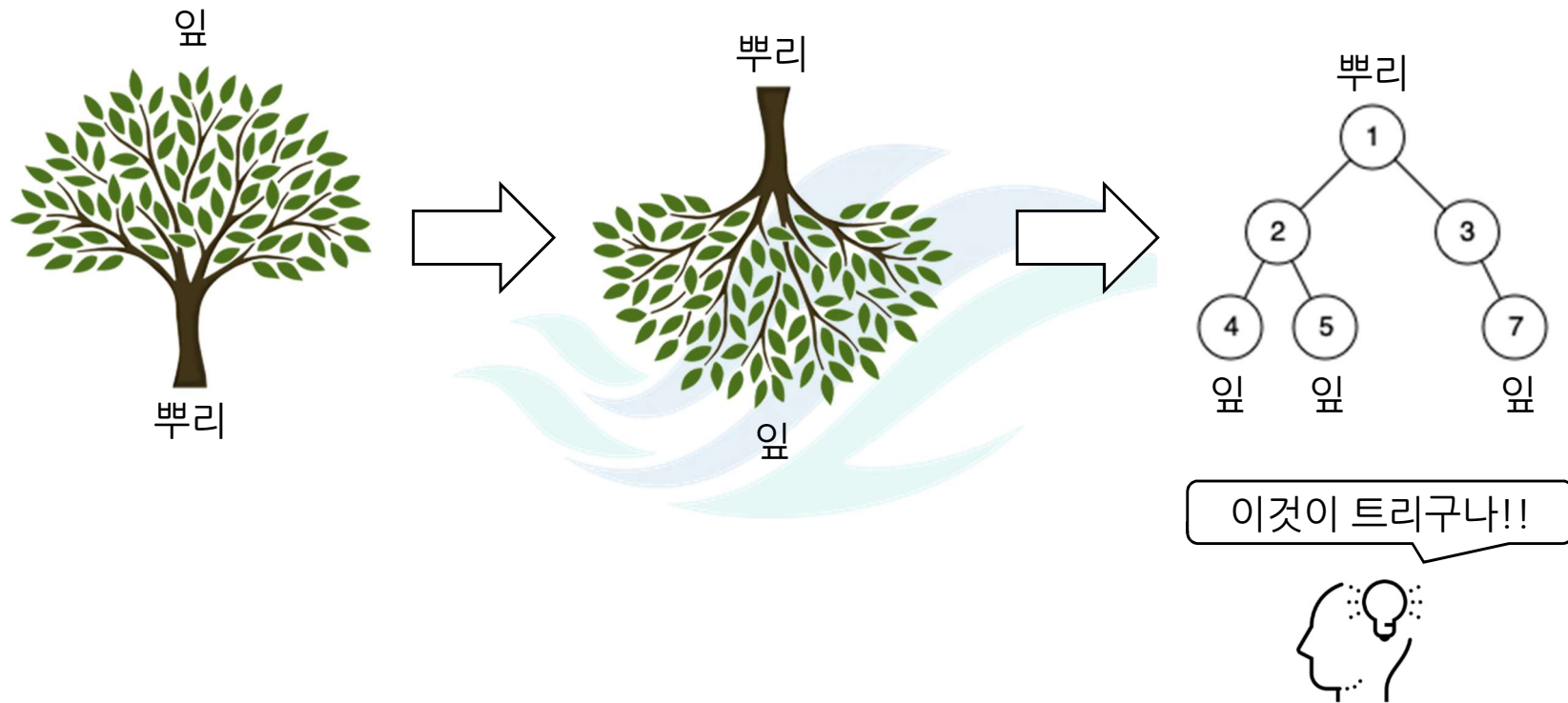


자료구조

한림대학교
소프트웨어융합대학
김태운

목차

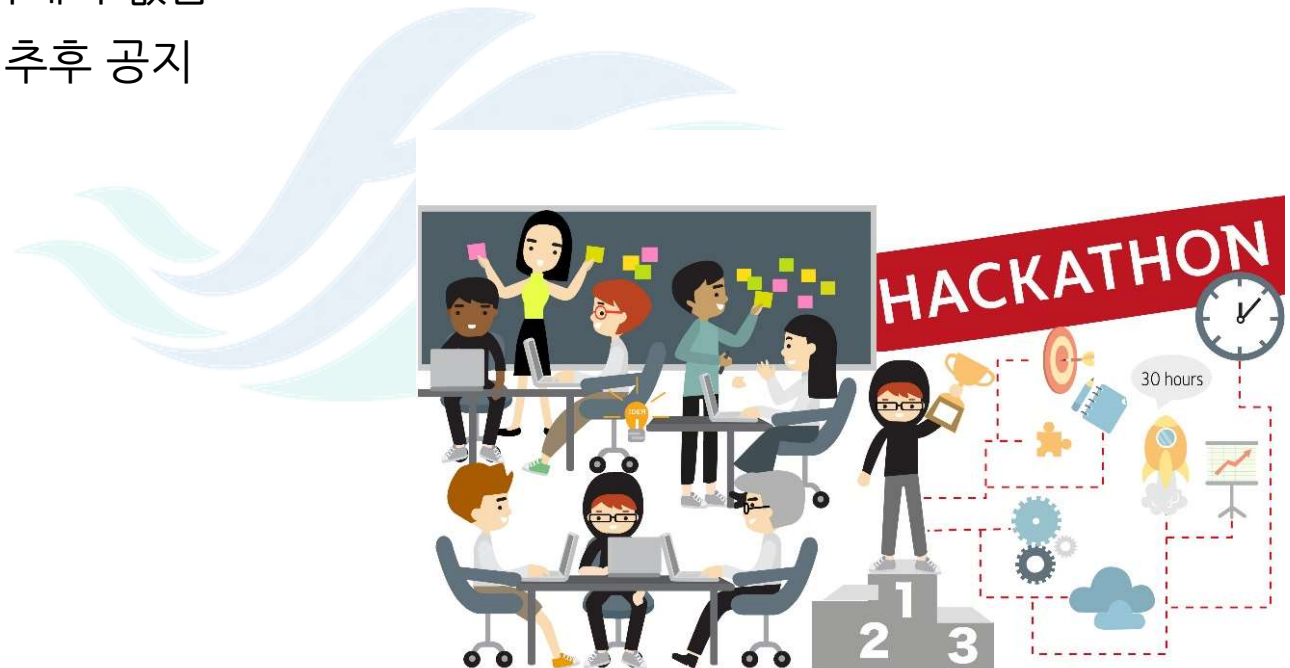
- 트리 (Tree)



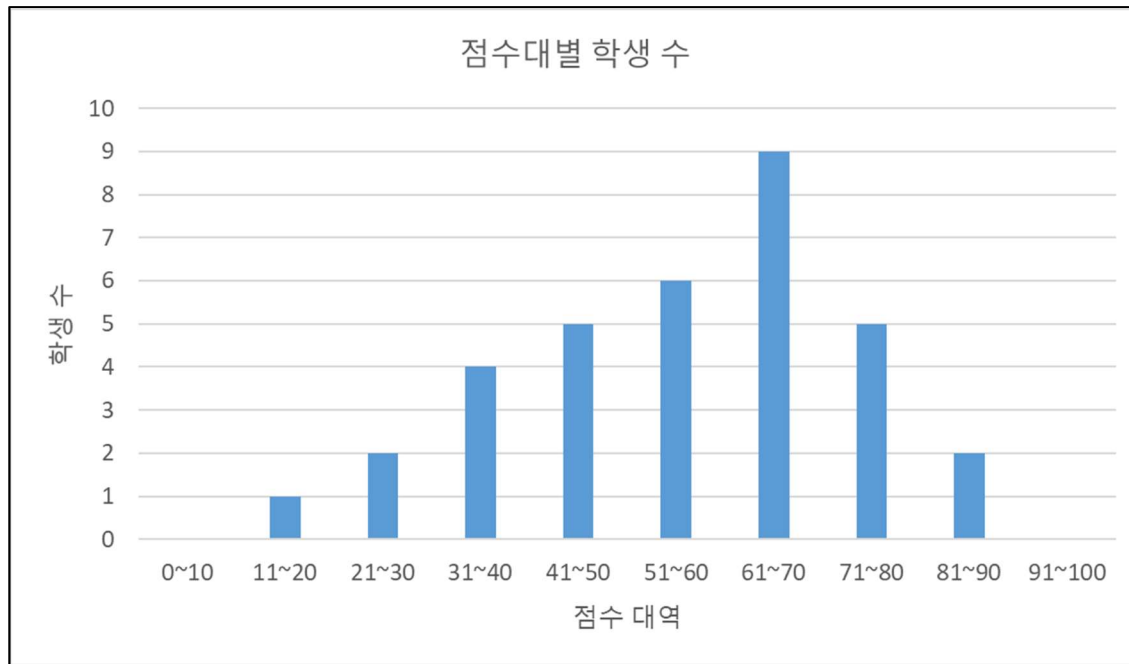
공지

- 해커톤

- 11 또는 12월 중으로 실습 시간에 해커톤을 진행하며, 결과는 성적에 반영됨
(해커톤 점수 = 실습과제 1개 점수)
- 해당 주차에는 실습과제가 없음
- 진행 일정 및 방법은 추후 공지
- 수상 혜택
 - 1등 : 큰 칭찬
 - 2등 : 칭찬
 - 3등 : 격려



중간고사 통계



평균	56.26
최고점	87

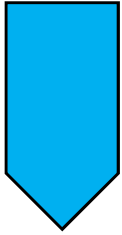
채점 결과를 확인하고 싶은 학생은
개별적으로 연락 바랍니다.
(taewoon@hallym.ac.kr,
공학관 1344)



중간고사

중간고사 문제 중에서 질문?





트리 (Tree)



순차적 자료구조의 단점

- 배열이나 연결리스트 (\Rightarrow 순차적 자료구조) : 데이터를 일렬로 저장/접근하기 때문에 탐색 연산이 순차적으로 수행되는 단점
 - 배열은 접근시간이 $O(1)$ 이지만, 탐색은 여전히 $O(N)$
 - 연결 리스트는 접근/탐색 모두 $O(N)$
- 배열은 미리 정렬해 놓으면 이진탐색을 통해 효율적인 탐색이 가능하지만, 삽입이나 삭제 후에도 정렬 상태를 유지해야 하므로 **삽입이나 삭제하는데 $O(N)$ 시간 소요**

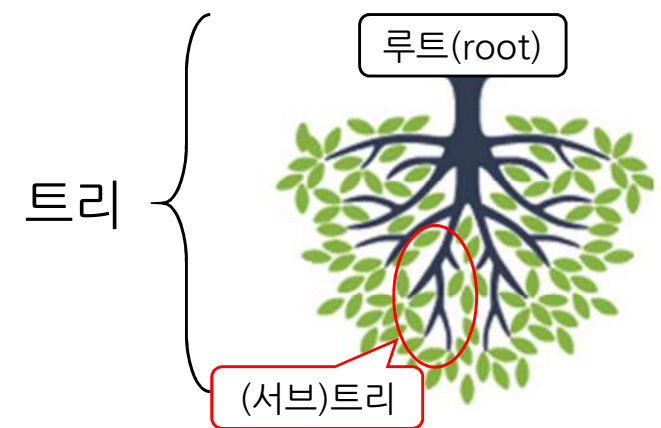
이진 탐색(Binary Search)의
시간 복잡도 : $O(\log N)$

해결책 : 계층적 자료구조?? 음... 트리?? 트리!!

트리 : 응용분야

- 트리 자료구조의 응용분야
 - 조직이나 기관의 계층구조
 - 컴퓨터 운영체제의 파일 시스템 (계층구조)
 - 자바 클래스 계층구조 등
- 트리는 탐색트리(Search Tree), 힙(Heap) 자료구조, 컴파일러의 수식을 위한 구문트리(Syntax Tree) 등...의 기본이 되는 자료구조로서 광범위하게 응용
- 트리는 일반적인 트리과 이진트리(Binary Tree)로 구분

트리



- 일반적인 트리(General Tree)는 실제 트리를 거꾸로 세워 놓은 형태의 자료구조
- HTML/XML 의 문서 트리, 자바 클래스 계층구조, 운영체제 파일시스템, 탐색 트리, 이항(Binomial)힙, 피보나치(Fibonacci)힙과 같은 우선순위큐에서 사용

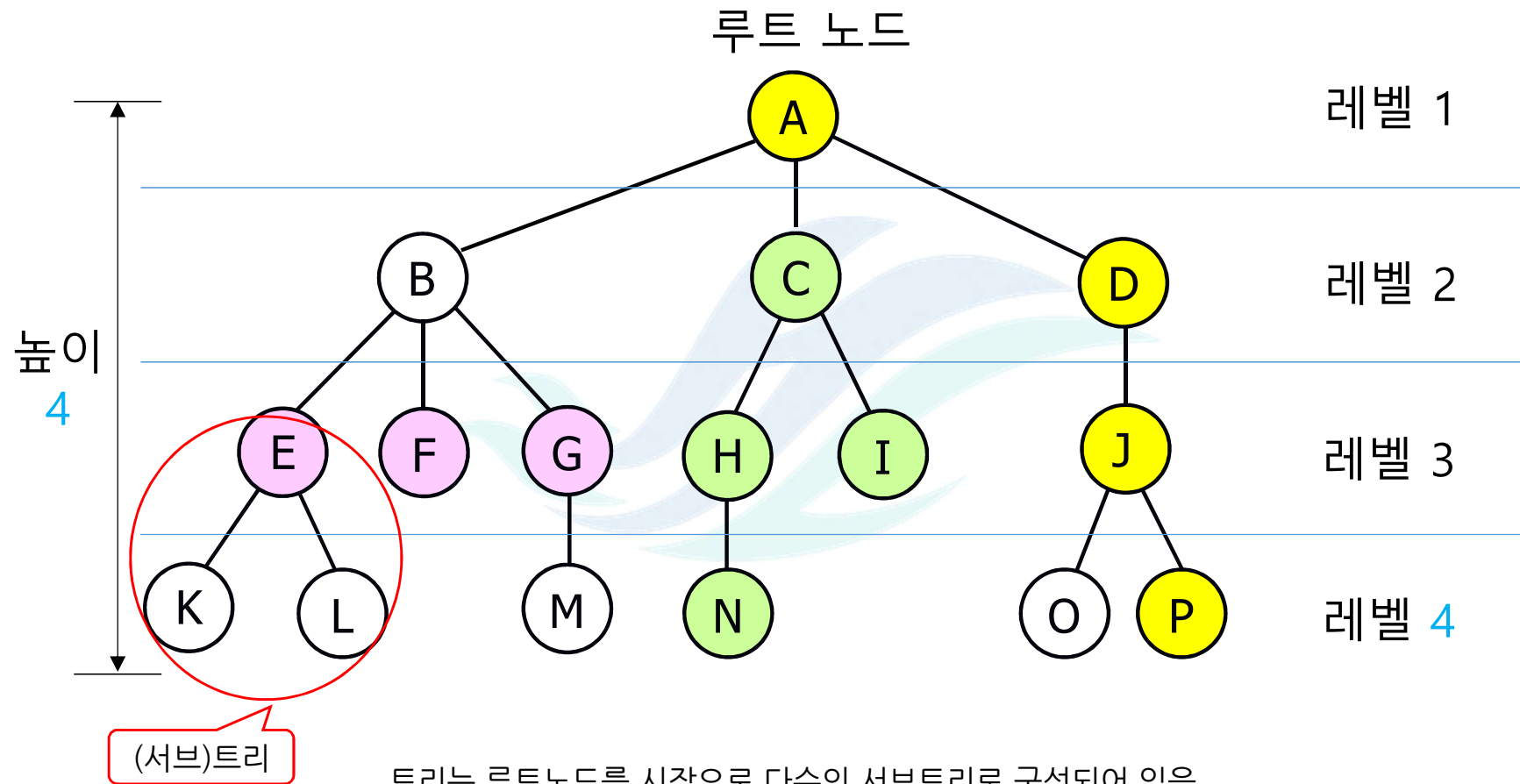
• 일반적인 트리의 정의

트리는 empty이거나, empty가 아니면 루트(root) 노드 R과 트리의 집합으로 구성되는데 각 트리의 루트노드는 R의 자식노드이다. 단, 트리의 집합은 공집합일 수도 있다

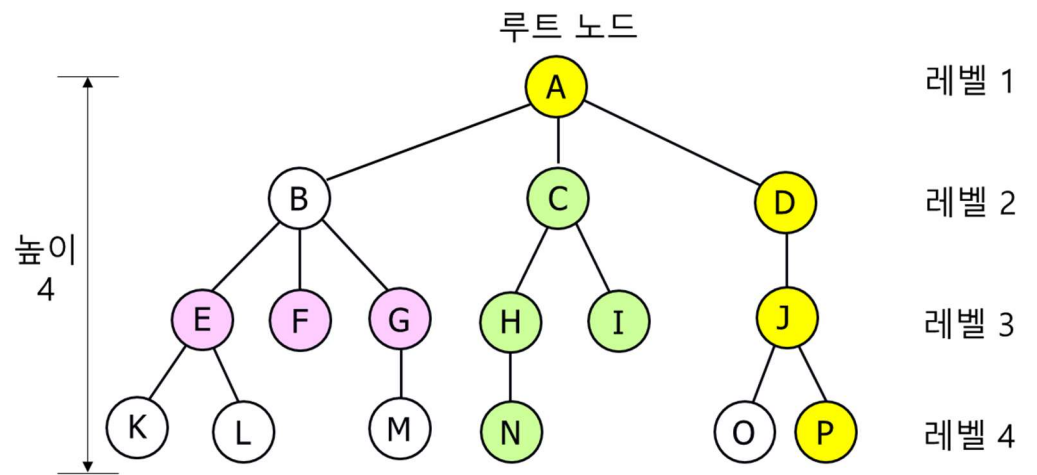
트리 : 용어

- 루트(Root) 노드: 트리의 최상위에 있는 노드
- 자식(Child) 노드: 노드 하위에 연결된 노드
- 차수(Degree): 자식노드 수
- 부모(Parent) 노드: 노드의 상위에 연결된 노드
- 이파리(Leaf) 노드: 자식이 없는 노드 (= 터미널 노드, 단말 노드)
- 형제(Sibling) 노드: 동일한 부모를 가지는 노드
- 조상(Ancessor) 노드: 루트노드까지의 경로상에 있는 모든 노드들의 집합
- 후손(Descendant) 노드: 노드 아래로 매달린 모든 노드들의 집합
- 서브트리(Subtree): 노드 자신과 후손 노드로 구성된 트리
- 레벨(Level): 루트 노드는 레벨 1, 아래 층/단계로 내려가며 레벨이 1씩 증가
 - 레벨은 깊이(Depth)와 같다.
- 높이(Height): 트리의 최대 레벨
- 키(Key): 탐색에 사용되는 노드에 저장된 정보

트리 : 구조



트리 : 구조/용어

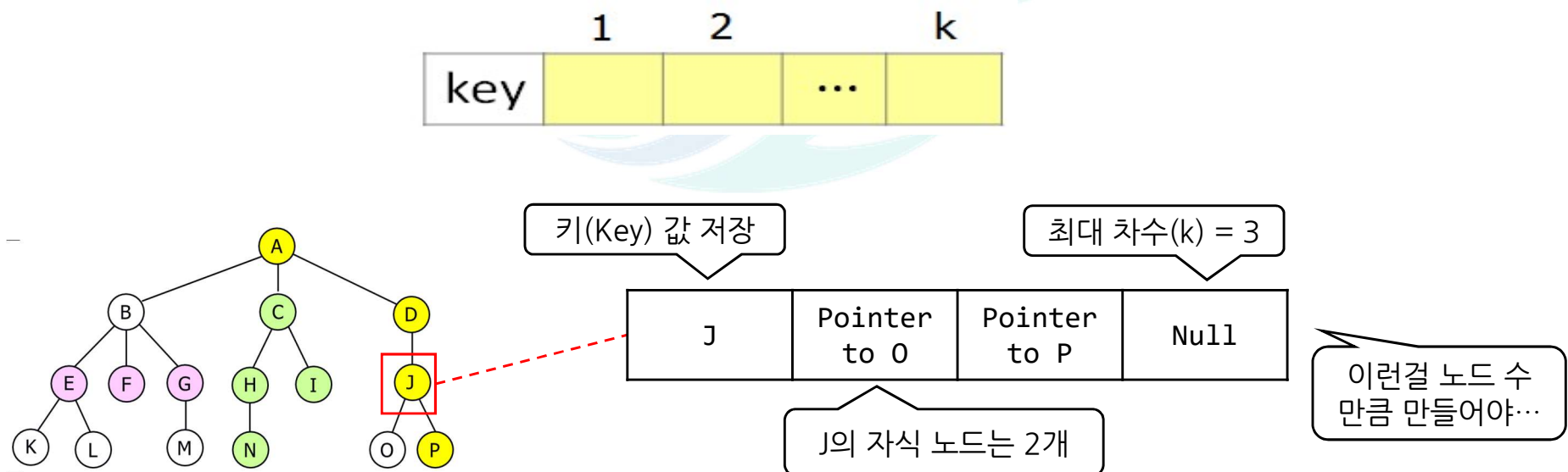


- A: 트리의 루트 노드
- B, C, D: A의 자식 노드 (= 부모가 같음)
- A의 차수: 3 (= 자식 노드 수)
- B, C, D의 부모 노드: A
- K, L, F, M, N, I, O, P: 이파리 노드들 (= 자식이 없는 노드)
- E, F, G의 부모가 B로 모두 같으므로 이들은 서로 형제 노드
- {B, C, D}, {H, I}, {K, L}, {O, P}도 각각 서로 형제 노드들
- C의 자손: {H, I, N}
- C를 루트 노드로 하는 서브 트리는 C와 C의 자손노드들로 구성된 트리
- P의 조상 노드: {J, D, A}
- 트리 높이: 4 (= 최대 레벨)

트리 안에 있는 트리

트리 : 표현 방법 (단순 배열)

- 이파리 노드(Leaf Node): 단말(Terminal)노드 또는 외부(External)노드
- 이파리가 아닌 노드: 내부(Internal)노드 또는 비 단말(Non-Terminal)노드
- 일반적인 트리를 메모리에 저장하는 일반적인 방법...
 - 각 노드에 키와 자식 수만큼의 레퍼런스를 저장
- 노드의 최대 차수가 k라면, k개의 레퍼런스 필드를 다음과 같이 선언 해야함:

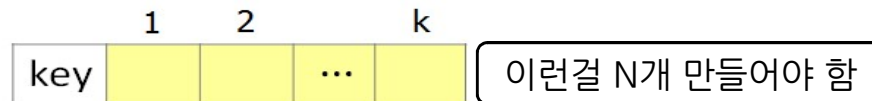


트리 : 표현 방법 (단순 배열)

- N개의 노드가 있는, 최대 차수가 k인 트리

$$\text{null 레퍼런스 수} = N*k - (N-1) = N(k-1) + 1$$

$N*k$ = 총 레퍼런스의 수



$(N-1)$ = 트리에서 부모-자식을 연결하는 레퍼런스 수(=null 아닌 레퍼런스 수)

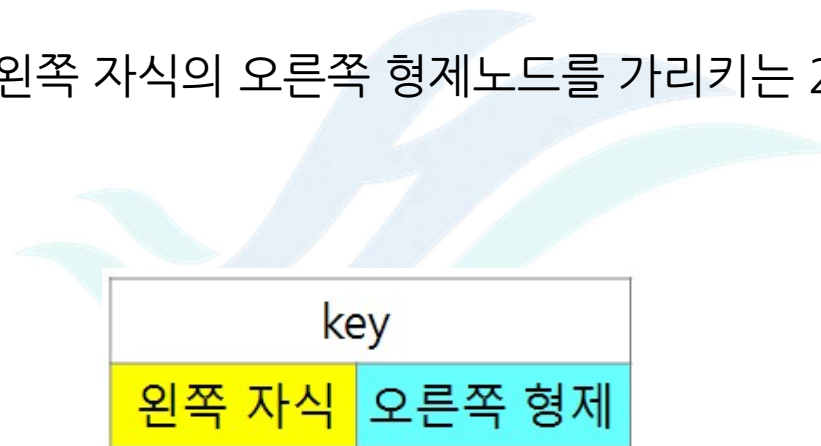
노드가 N개 있으면, 트리를 나타내기 위해 N-1개의 레퍼런스만 있으면 된다. 즉, N-1개의 레퍼런스만 실제로 사용되고, 나머지 레퍼런스는 null 값이 된다.

- k가 클수록 **메모리의 낭비**가 심해지는 것은 물론 트리를 탐색하는 과정에서 null 레퍼런스를 확인해야 하므로 **시간적으로도 매우 비효율적**

트리 : 표현 방법 (개선)

왼쪽자식-오른쪽형제 표현

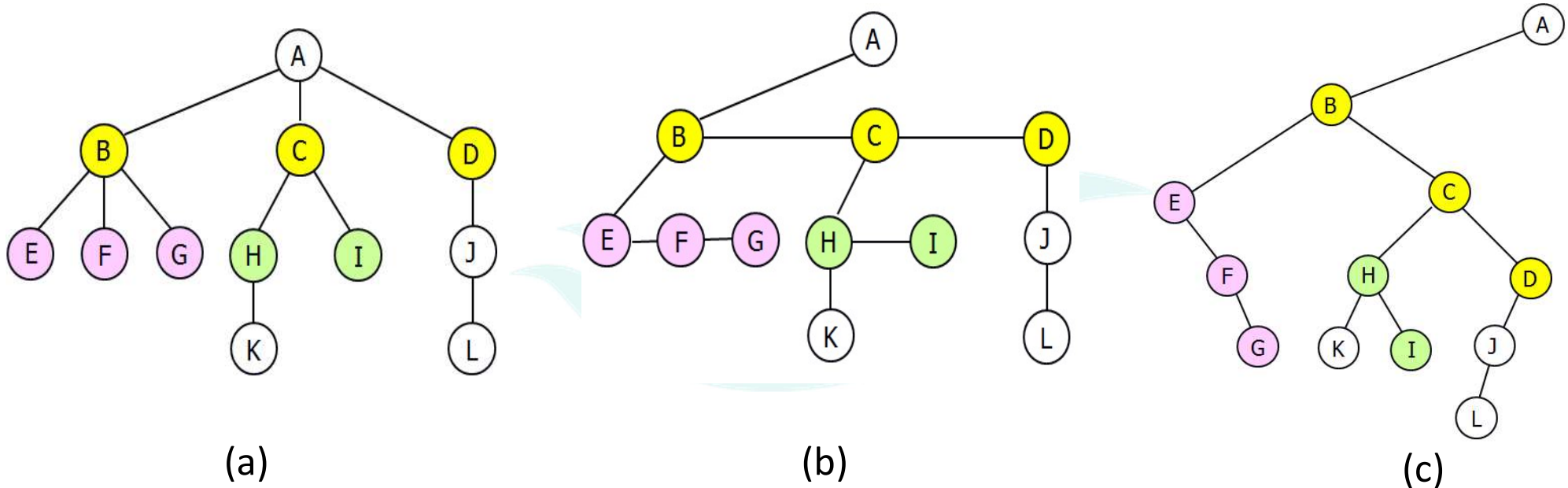
- 왼쪽자식-오른쪽형제(Left Child-Right Sibling) 표현
 - 직전의 단순배열 표현법의 비효율적인 측면을 개선한 표현법
 - 노드의 왼쪽 자식과 왼쪽 자식의 오른쪽 형제노드를 가리키는 2개의 레퍼런스만을 사용



key	
왼쪽 자식	오른쪽 형제

트리 : 왼쪽자식-오른쪽형제 표현

- [예제] (a)의 트리를 왼쪽자식-오른쪽형제 표현으로 변환하면, (b)의 트리를 얻으며, (c)는 (b)의 트리를 45도 시계 방향으로 회전시킨 것



예시:

A	
B	C

E	
Null	F

L	
Null	Null

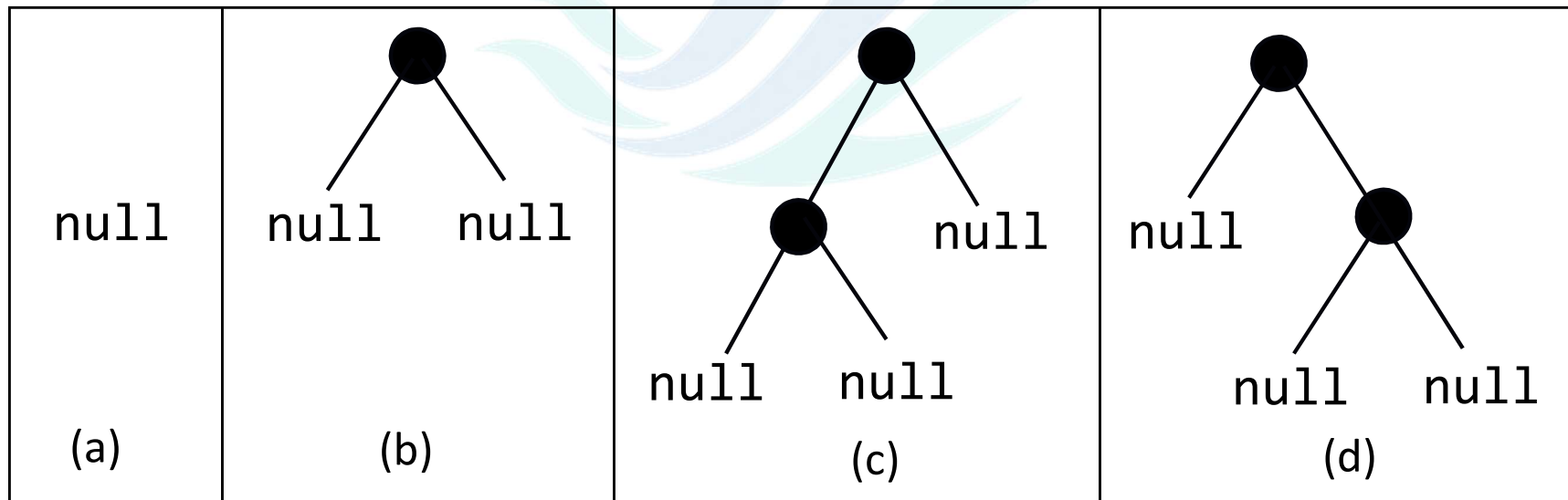
이진트리

- 이진트리(Binary Tree): 각 노드의 자식 수가 2이하인 트리
- 컴퓨터 분야에서 널리 활용되는 자료구조
 - 이진트리는 데이터의 구조적인 관계를 잘 반영하고,
 - 효율적인 삽입과 탐색을 가능하게 하며,
 - 이진트리의 서브트리를 다른 이진트리의 서브트리와 교환하는 것이 쉽다는 장점
- 이진트리에 대한 용어는 일반적인 트리에 대한 용어와 동일 (다음 페이지...)

이진트리

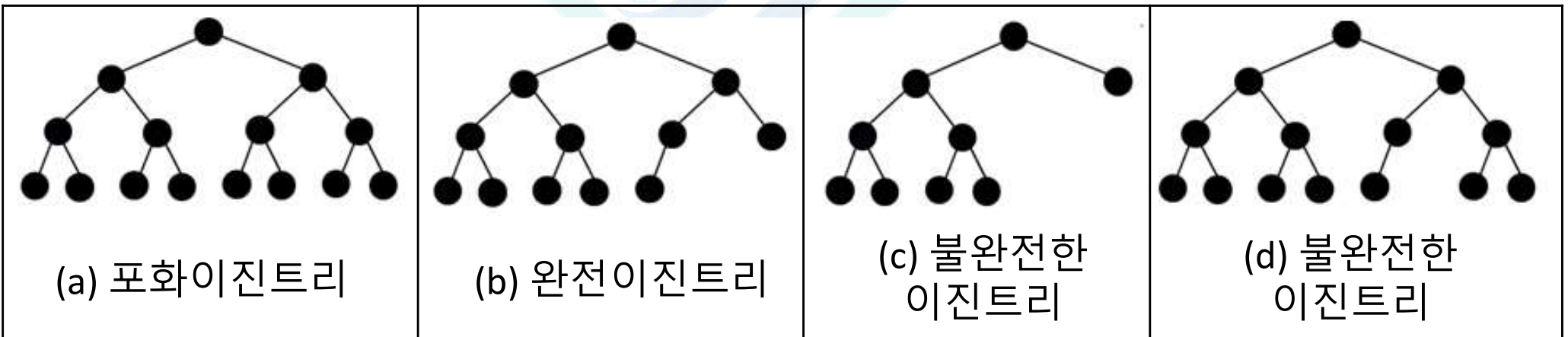
[정의] 이진트리는 empty이거나, empty가 아니면, 루트노드와 2개의 이진트리인 왼쪽 서브트리와 오른쪽 서브트리로 구성된다.

- (a) empty인(= 비어있는) 트리
- (b) 루트노드만 있는 이진트리
- (c) 루트노드의 오른쪽 서브트리가 없는(empty) 이진트리
- (d) 루트노드의 왼쪽 서브트리가 없는 이진트리



이진트리 : 특별한 형태

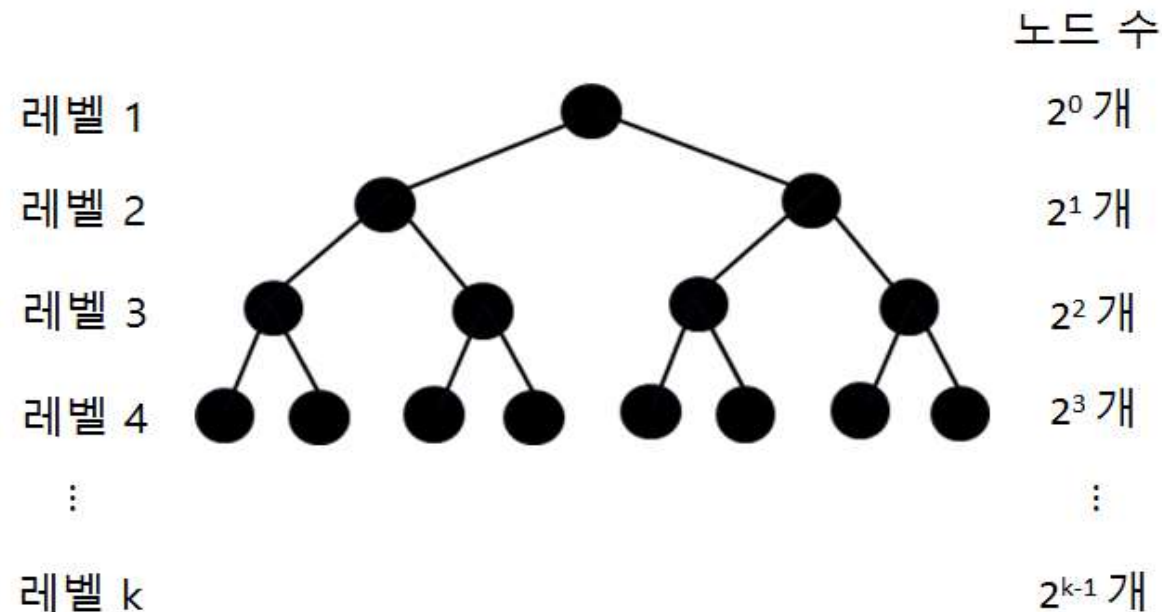
- 포화이진트리(Full Binary Tree):
 - 모든 이파리노드의 깊이가 같고, 각 내부노드가 2개의 자식노드를 가지는 트리
- 완전이진트리(Complete Binary Tree):
 - 마지막 레벨을 제외한 각 레벨이 노드들로 꽉 차있고,
 - 마지막 레벨에는 노드들이 왼쪽부터 빠짐없이 채워진 트리
- 포화이진트리는 완전이진트리이다.



이진트리 : 속성

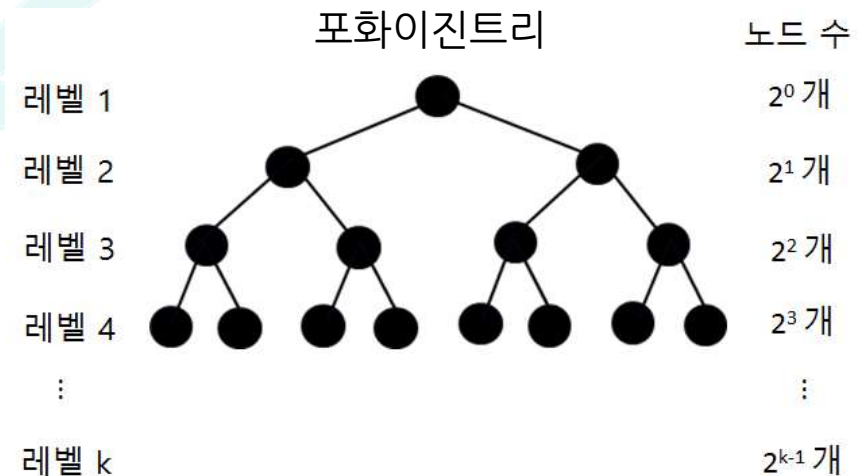
- 레벨 k 에 있는 최대 노드 수 = 2^{k-1} , $k = 1, 2, 3, \dots$
- 높이가 h 인 포화이진트리에 있는 노드 수 = $2^h - 1$
- N 개의 노드를 가진 완전이진트리의 높이 = $\lceil \log_2(N+1) \rceil$

Floor 함수
(소수점 올림 연산)



100

- 에 있는 노드 수
- $$2^{h-1} = 2^h - 1$$
- $$= \log_2(N+1)$$
- 레벨 1



이진트리 : 속성

- N개의 노드를 가진 완전이진트리에서 N이 $2^h - 1$ 보다 클 수 없으므로,

$$\text{높이 } h = \lceil \log_2(N+1) \rceil$$

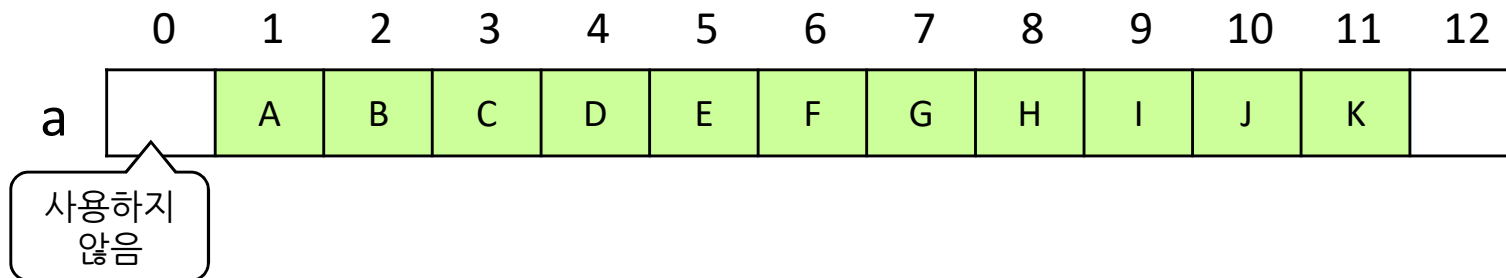
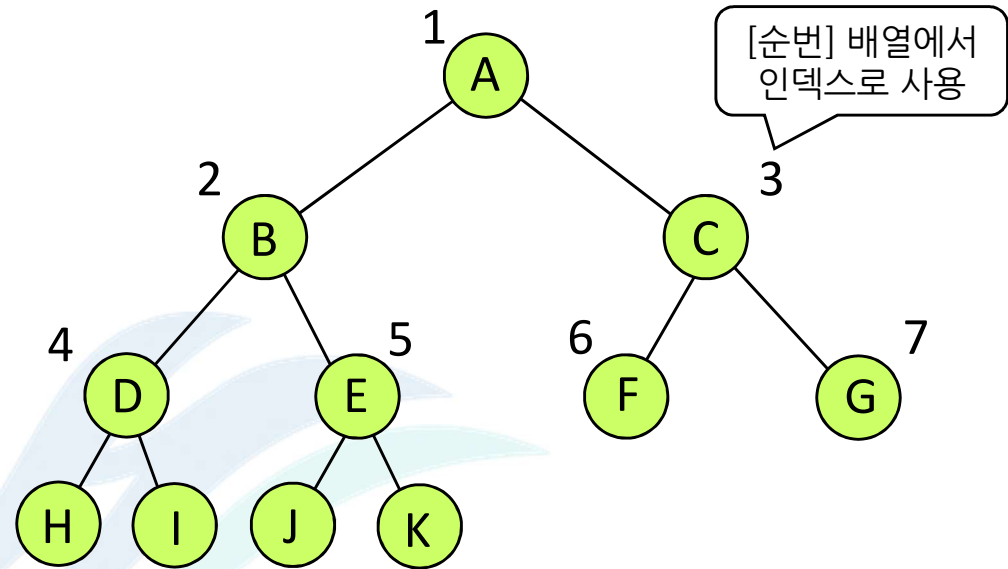
- 높이가 h인 완전이진트리에 존재 할 수 있는 최대 노드 수

$$2^{h-1} \sim 2^h - 1$$

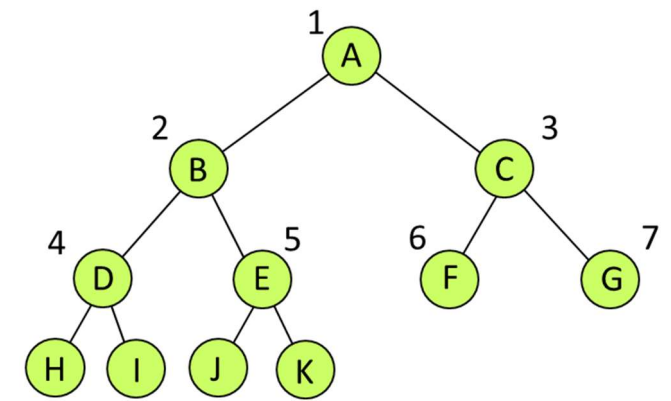
- 노드 수가 2^{h-1} 보다 작으면 높이는 $(h - 1)$ 가 됨
- $2^h - 1$ 보다 크면 높이는 $(h + 1)$ 이 됨

이진트리 : 배열

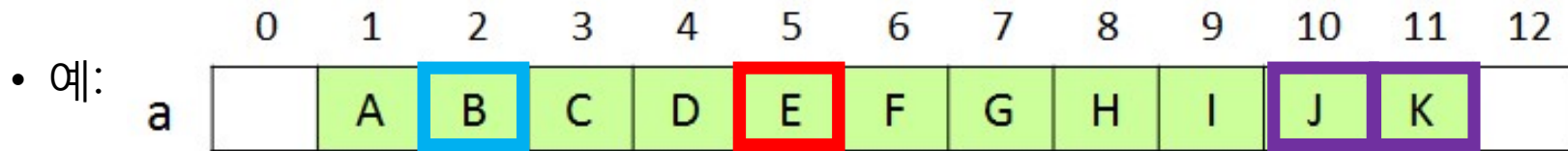
- 이진트리를 배열에 저장하는 방법
 - 각 노드의 위치에 순번을 정하고 순번을 인덱스로 사용하여 노드를 배열에 저장



이진트리 : 배열



- 배열에 저장하면 노드의 부모노드와 자식노드가 배열의 어디에 저장되어 있는지를 다음과 같은 규칙을 통해 쉽게 알 수 있다. 단, 트리에 N개의 노드가 있다고 가정
 - $a[i]$ 의 부모노드는 $a[i/2]$ 에 있다. 단, $i > 1$ 이다.
 - $a[i]$ 의 왼쪽 자식노드는 $a[2i]$ 에 있다. 단, $2i \leq N$ 이다.
 - $a[i]$ 의 오른쪽 자식노드는 $a[2i+1]$ 에 있다. 단, $2i + 1 \leq N$ 이다.

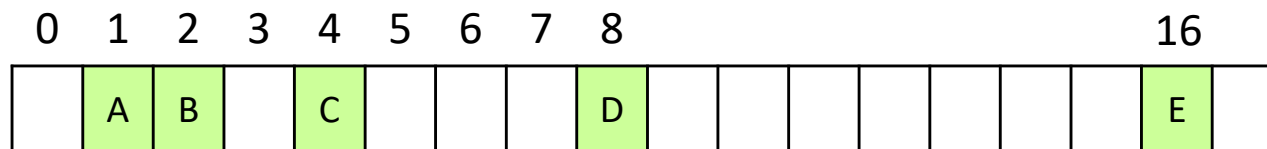
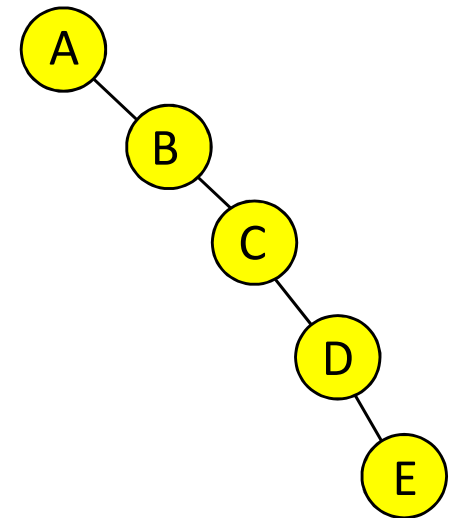
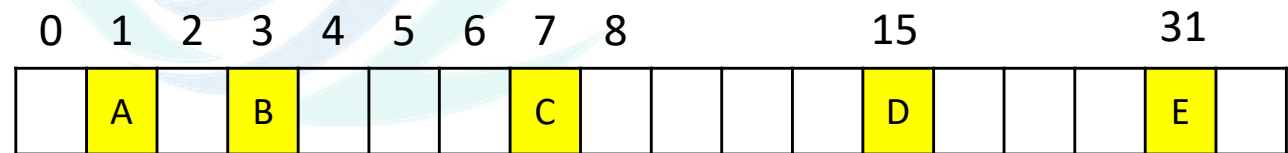
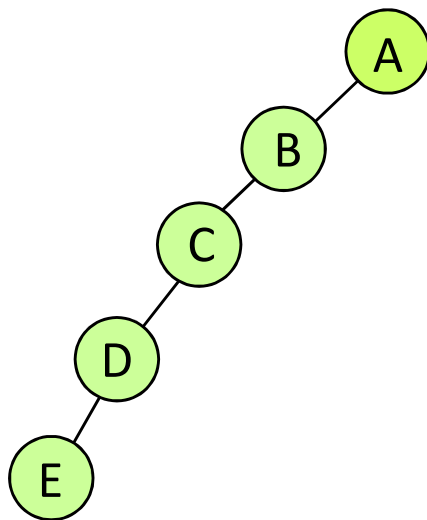


- E의 부모노드는 $a[5/2] = a[2]$ 에 있는 B
- E의 왼쪽과 오른쪽 자식은 각각 $a[2 \times 5] = a[10]$ 과 $a[2 \times 5 + 1] = a[11]$ 에 저장된 J와 K

참고: Float/double 을 int로 type casting 시, 소수점 버림

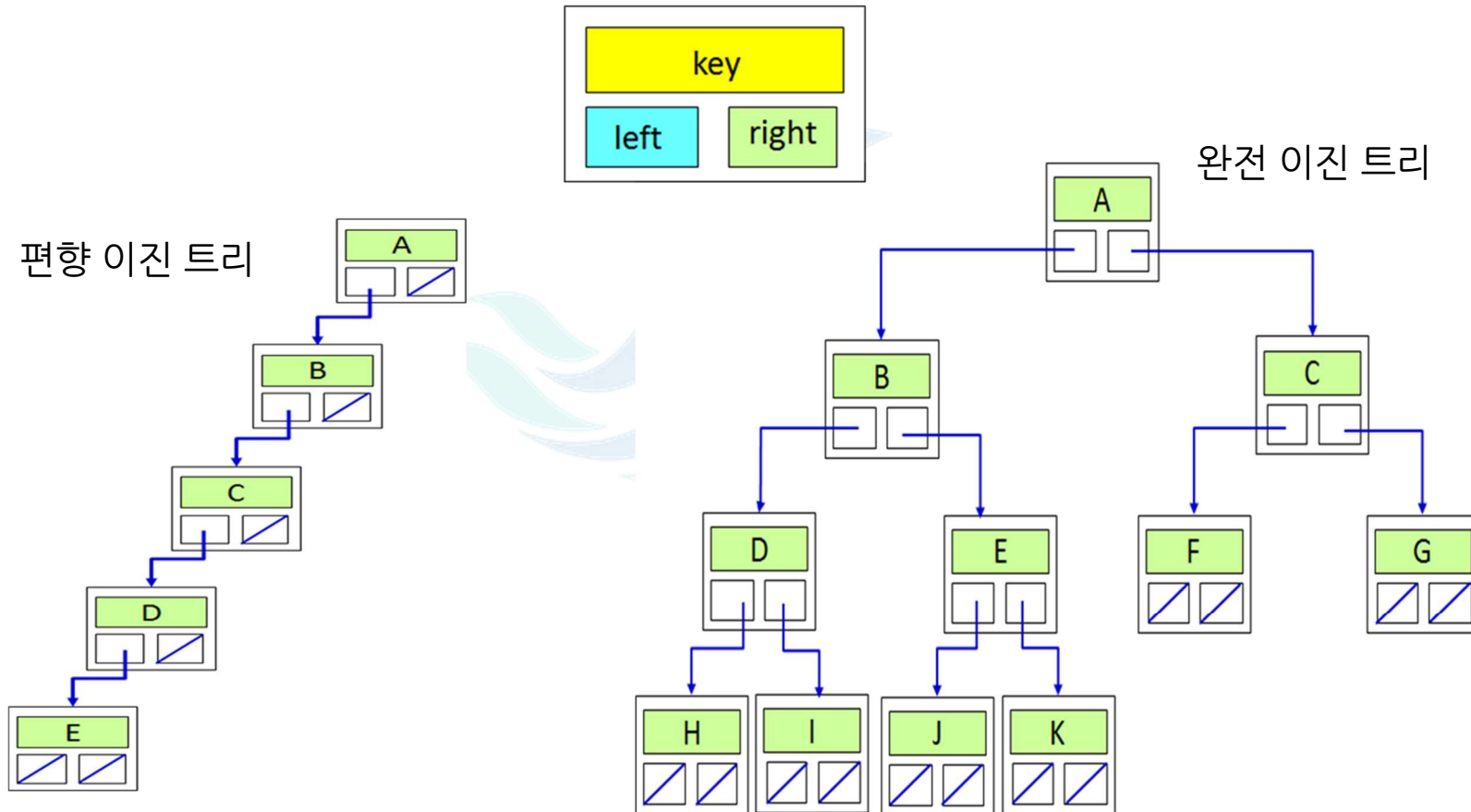
이진트리 : 배열, 편향이진트리

- 편향(Skewed)이진트리를 배열에 저장하는 경우, 트리의 높이가 커질 수록 메모리 낭비가 심화됨



이진트리 : 연결 리스트 표현 방법

- 노드는 키와 2개의 레퍼런스 필드, 즉, left와 right를 가진다. // 연결 리스트
 - 레퍼런스 필드에는 자식 노드를 가리키는 레퍼런스 저장



이진트리를 위한 Node 클래스

```
01 public class Node<Key extends Comparable<Key>> {  
02     private Key item;  
03     private Node<Key> left;  
04     private Node<Key> right;  
05     public Node( Key newItem, Node lt, Node rt ) { // 노드 생성자  
06         item = newItem; left = lt; right = rt; }  
07     public Key getKey( ) { return item; }  
08     public Node<Key> getLeft( ) { return left; }  
09     public Node<Key> getRight( ) { return right; }  
10     public void setKey(Key newItem) { item = newItem;}  
11     public void setLeft(Node<Key> lt) { left = lt;}  
12     public void setRight(Node<Key> rt) { right = rt;}  
13 }
```

public int compareTo(Key other) 메소드를 통해
2개의 키를 비교하기 위해 사용

- Line 01: Key를 generic 타입으로 사용. Key는 데이터(item)를 노드에 저장하는 목적.
- Line 05~06: 생성자
- Line 07~12: get, set 메소드

이진트리(BinaryTree) 클래스

```
01 import java.util.*;
02 public class BinaryTree<Key extends Comparable<Key>> {
03     private Node root;
04     public BinaryTree( ) { root = null; } // 트리 생성자
05     public Node getRoot( ) { return root; }
06     public void setRoot(Node newRoot) { root = newRoot; }
07     public boolean isEmpty( ) { return root == null; }
    // preorder(), inorder(), postorder(), levelorder(),
    // size(), height(), isEqual() 메소드 선언
}
```

- Line 04 : BinaryTree 클래스 생성자
- Line 05: root를 리턴
- Line 06: 트리의 루트 노드를 newRoot로 교체
- Line 07: 트리가 empty인지를 체크
- 나머지 메소드: 이진트리를 네 종류의 방식으로 순회하는 메소드와 기타 기본 연산을 위한 메소드 (자세한 내용은 다음 페이지 참고)

이진트리 : 순회

- 이진트리에서 수행하는 기본 연산은 트리를 순회(Traversal)하는 연산을 이용해서 수행할 수 있다.
 - 순회 : 노드를 따라가며 이동하는 것
- 이진트리의 4가지 순회하는 방식
 - 전위순회(Pre-order Traversal)
 - 중위순회(In-order Traversal)
 - 후위순회(Post-order Traversal)
 - 레벨순회(Level-order Traversal)

(방식은 각각 다르지만, 순회는 항상 트리의 루트노드부터 시작)

이진트리 : 순회

- 전위, 중위, 후위순회는 트리를 순회하는 중에 **노드를 방문하는 시점에 따라** 구분된다.
 - 방문 = 확인.
 - 예(**중위** 순위): 노드 A를 기준으로...노드의 왼쪽 자식 확인 => **노드에 저장된 키 값 확인** => 노드의 오른쪽 자식 확인 // 가리키는 노드를 “중간” 시점에 방문
- 전위, 중위, 후위순회는 루트노드로부터 정해진 순서로 이진트리의 노드들을 지나가는데, **특정 노드에 도착하자마자 그 노드를 방문하는지, 일단 지나치고 나중에 방문하는지에 따라** 구분됨

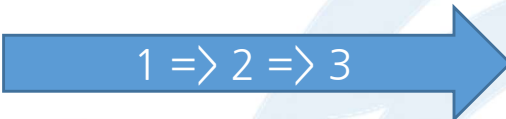
이진트리 : 순회

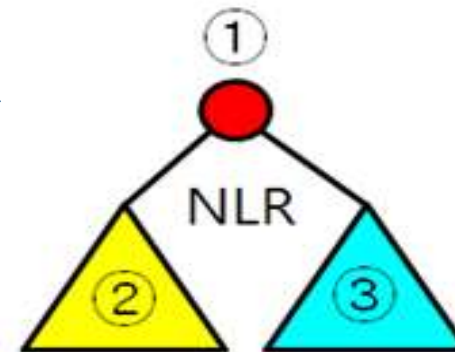
- 집을 노드라고 하면, 노드를 방문하는 것은 문을 열고 집안에 들어가는 것
- 사람이 노드(집)에는 도착했으나 집을 방문하는 것을 나중에 미루고 왼쪽이나 오른쪽 길로 다른 집을 찾아 나설 수도 있음



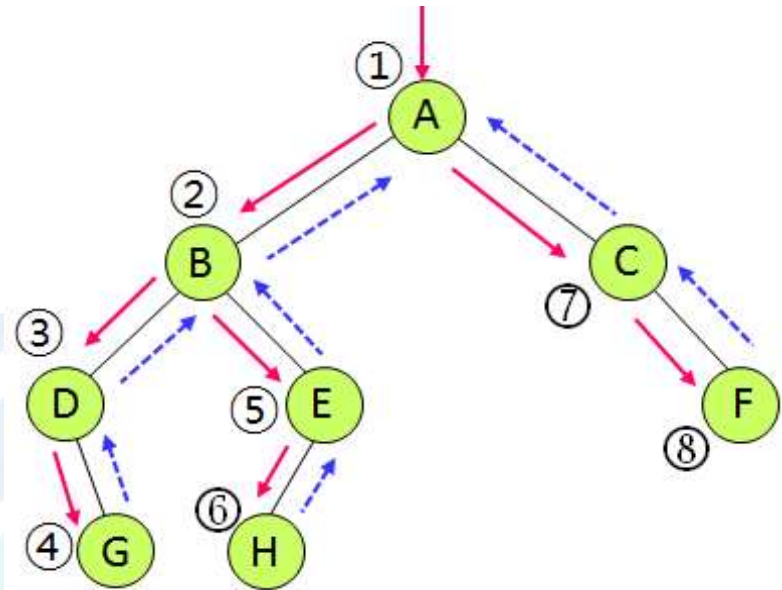
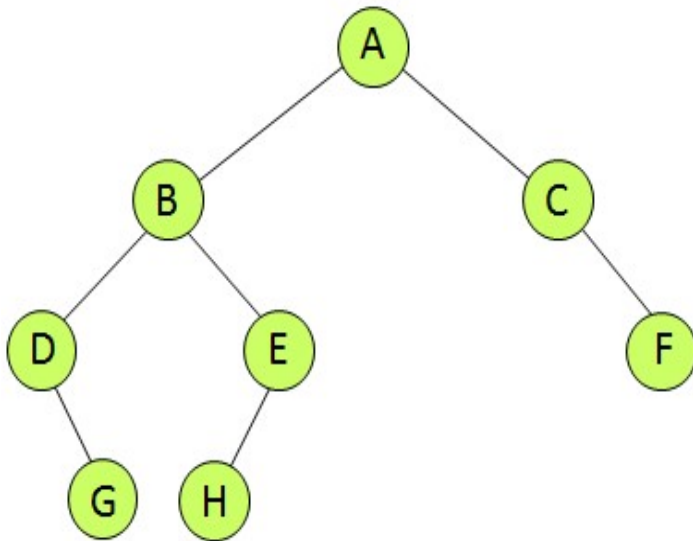
- 모든 순회 방식은 루트노드로부터 순회를 시작하여 **트리의 모든 노드를 반드시 1 번씩 방문**해야 순회가 종료됨

이진트리 : 전위순회 (노드>좌>우)

- 전위순회는 노드 x에 도착했을 때 x를 먼저 방문
- 그 다음에 x의 왼쪽 자식노드로 순회를 계속
- x의 왼쪽 서브트리의 모든 노드를 방문한 후에는 x의 오른쪽 서브트리의 모든 후손 노드 방문
- 전위순회의 방문 규칙: 
- 각 서브트리의 방문은 동일한 방식으로
- 전위순회 순서를 NLR 또는 VLR로 표현
 - 여기서 N은 노드(Node)를 방문한다는 뜻이고, V는 Visit(방문)을 의미
 - L은 왼쪽, R은 오른쪽 서브트리로 순회를 진행한다는 뜻



이진트리 : 전위순회 (노드>좌>우)



- 실선 화살표를 따라서 A, B, D, G, E, H, C, F 순으로 방문
- 점선 화살표는 노드의 서브트리에 있는 모든 노드들을 방문한 후에 부모노드로 복귀
- 복귀하는 것은 프로그램에서 메소드 호출이 완료된 후에 리턴하는 것과 같음
- 단, 노드를 방문 하는 것은 노드의 key를 출력 한다고 가정

이진트리 : 전위순회 (노드>좌>우)

```
01 public void preorder(Node n) { // 전위순회
02     if (n != null) {
03         System.out.print(n.getKey()+" "); // 노드 n 방문
04         preorder(n.getLeft()); // n의 왼쪽 서브트리를 순회하기 위해
05         preorder(n.getRight()); // n의 오른쪽 서브트리를 순회하기 위해
06     }
07 }
```

방문

재귀호출

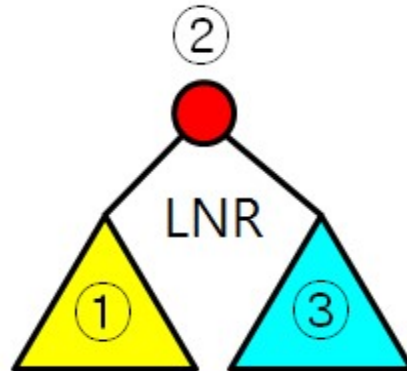
- 트리의 루트노드를 인자로 전달하여 호출
- Line 02: 노드 n이 null 인지 검사. Null이면 이전 호출된 곳으로 돌아가고(= 리턴), null이 아니면 line 03 에서 노드 n을 방문 후 재귀호출
- Line 04: 노드 n의 왼쪽 자식노드로 재귀호출하여 왼쪽 서브트리의 모든 노드들을 방문
- Line 05: 노드 n의 오른쪽 자식노드로 재귀호출하고 오른쪽 서브트리의 모든 노드들을 방문

[재귀함수]

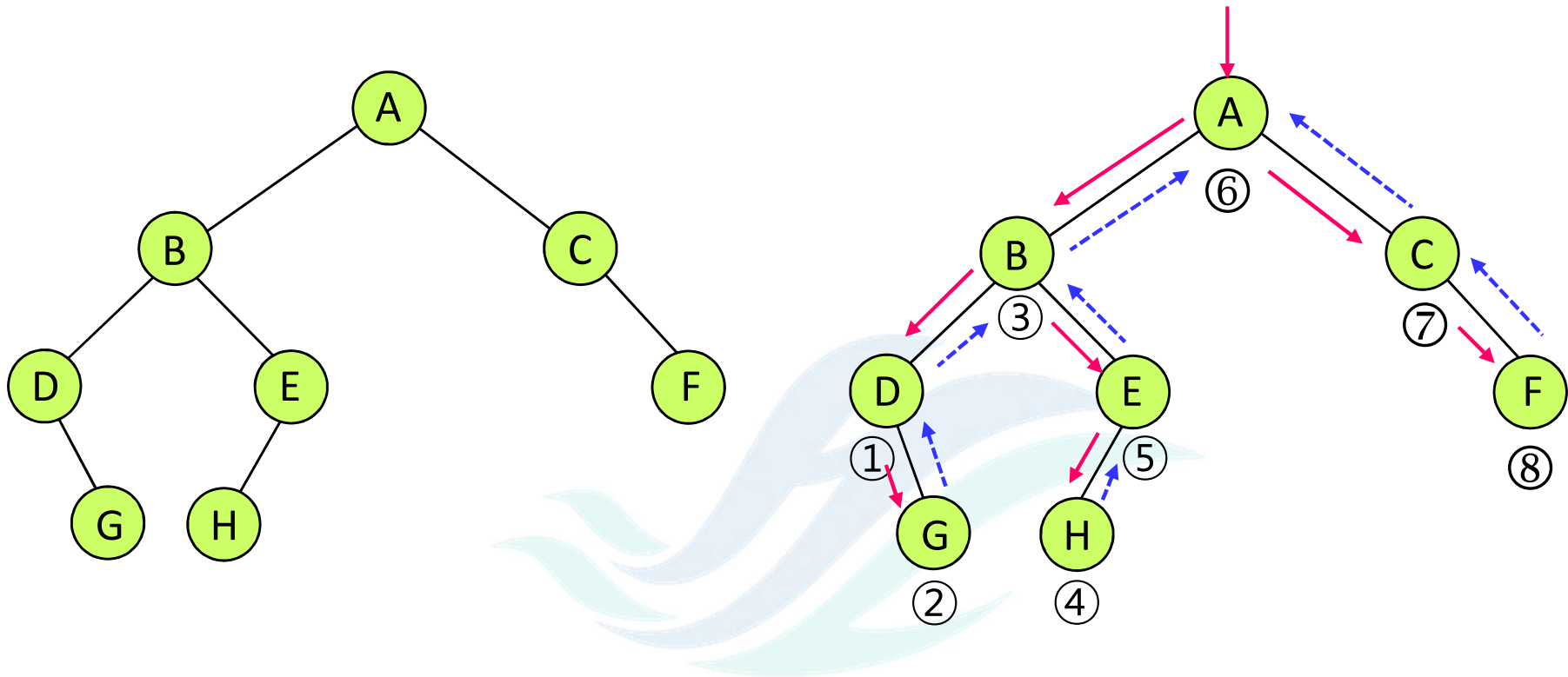
- 기본 케이스 : 재귀호출 안함 (if문 들어가지 못하고 리턴 함)
- 재귀호출 케이스 : 재귀호출 함 (if문에 들어가서 재귀호출 수행)

이진트리 : 중위순회 (좌>노드>우)

- 중위순회는 노드 x에 도착하면 x의 방문을 보류하고 x의 왼쪽 서브트리로 순회를 진행.
- 왼쪽 서브트리의 모든 노드들을 방문한 후에 x를 방문
- x를 방문한 후에는 x의 오른쪽 서브트리를 같은 방식으로 방문
- 중위순회 순서를 LNR 또는 LVR로 표현



이진트리 : 중위순회 (좌>노드>우)



중위순회: D, G, B, H, E, A, C, F 순으로 방문

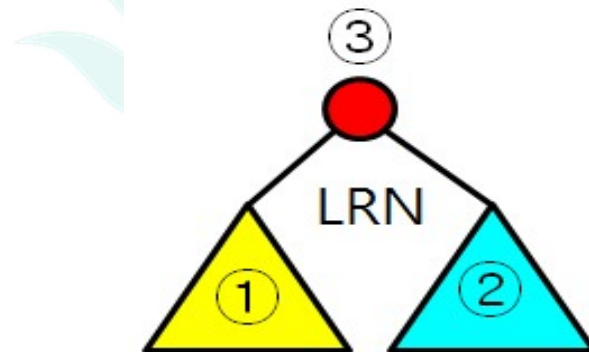
이진트리 : 중위순회 (좌>노드>우)

```
01 public void inorder(Node n){ // 중위순회
02     if (n != null) {
03         inorder(n.getLeft()); // n의 왼쪽 서브트리를 순회하기 위해
04         System.out.print(n.getKey()+" "); // 노드 n 방문
05         inorder(n.getRight()); // n의 오른쪽 서브트리를 순회하기 위해
06     }
07 }
```

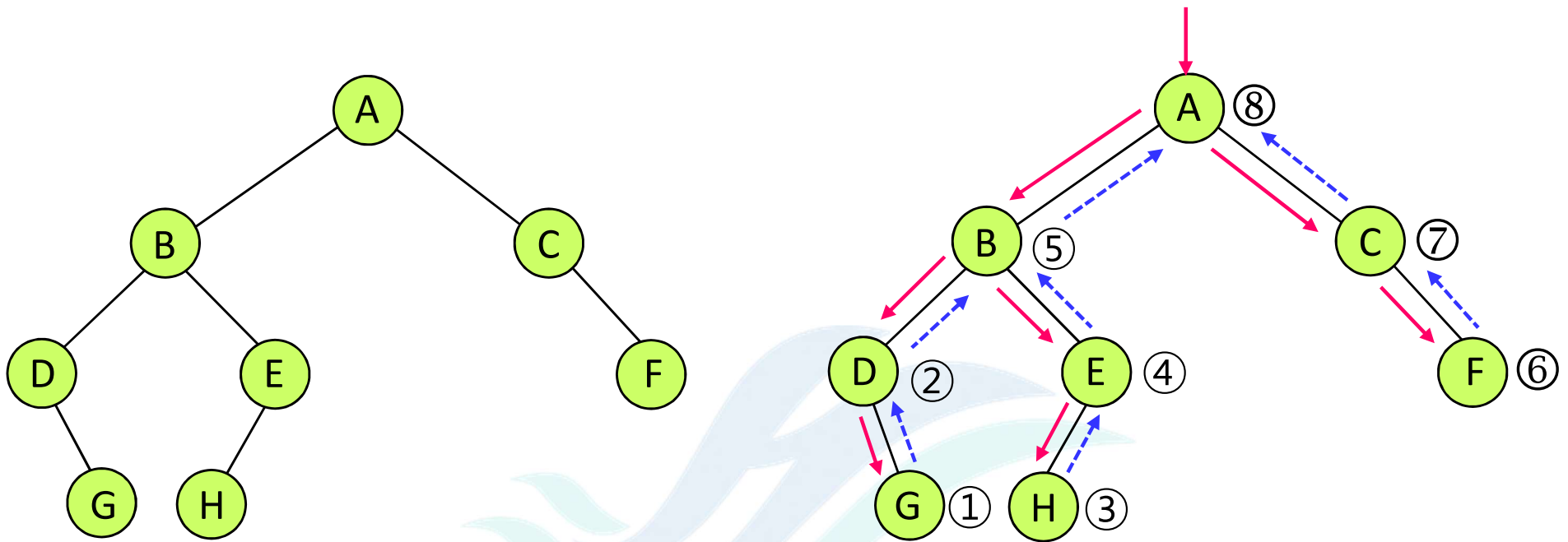
- 루트노드를 인자로 전달하여 호출
- Line 02: 노드 n이 null 인지를 검사. Null이면 이전 호출된 곳으로 돌아가고, null이 아니면 line 03으로 이동.
- Line 03: 노드 n의 왼쪽 자식노드로 재귀호출하여 왼쪽 서브트리의 모든 노드 방문
- Line 04: 노드 n을 방문
- Line 05: 노드 n의 오른쪽 자식노드로 재귀호출하고 오른쪽 서브트리의 모든 노드 방문

이진트리 : 후위순회 (좌>우>노드)

- 후위순회는 노드 x에 도착하면 x의 방문을 보류하고 x의 왼쪽 서브트리로 순회
- x의 왼쪽 서브트리를 방문한 후에는 x의 오른쪽 서브트리를 같은 방식으로 방문
- 마지막에 x를 방문
- 후위순회 순서를 LRN 또는 LRV로 표현



이진트리 : 후위순회 (좌>우>노드)



- 후위순회: G, D, H, E, B, F, C, A 순으로 방문

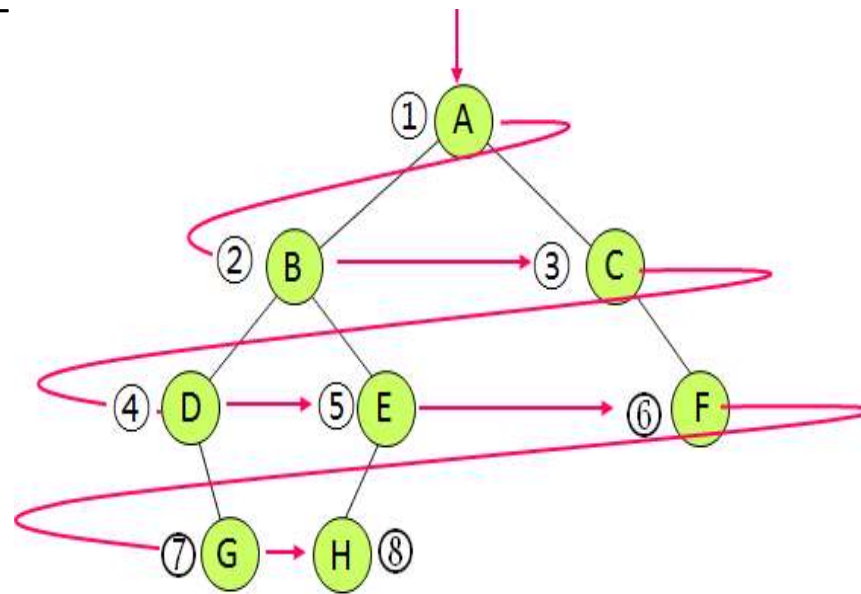
이진트리 : 후위순회 (좌>우>노드)

```
01 public void postorder(Node n) { // 후위순회
02     if (n != null) {
03         postorder(n.getLeft()); // n의 왼쪽 서브트리를 순회하기 위해
04         postorder(n.getRight()); // n의 오른쪽 서브트리를 순회하기 위해
05         System.out.print(n.getKey()+" "); // 노드 n 방문
06     }
07 }
```

- 트리의 루트노드를 인자로 전달하여 호출
- Line 02: 노드 n이 null 인지 검사. Null이면 리턴, null이 아니면 다음 줄로 이동
- Line 03: 노드 n의 왼쪽 자식노드로 재귀호출하여 왼쪽 서브트리의 모든 노드 방문
- Line 04: 노드 n의 오른쪽 자식노드로 재귀호출하고 오른쪽 서브트리의 모든 노드 방문
- 끝으로 line 05에서 노드 n을 방문

이진트리 : 레벨순회

- 레벨순회는 루트노드가 있는 최상위 레벨부터 시작하여 각 레벨마다 좌에서 우로 노드들을 방문



방문 순서 : A B C D E F G H

이진트리 : 레벨순회

import java.util.*;

```
01 public void levelorder(Node root) { // 레벨순회
02     Queue<Node> q = new LinkedList<Node>(); // 큐 자료구조 이용
03     Node t;
04     q.add(root); // 루트 노드 큐에 삽입
05     while (!q.isEmpty()) {
06         t = q.remove(); // 큐에서 가장 앞에 있는 노드 제거
07         System.out.print(t.getKey()+" "); // 제거된 노드 출력(방문)
08         if (t.getLeft() != null) // 제거된 왼쪽 자식이 null이 아니면
09             q.add(t.getLeft()); // 큐에 왼쪽 자식 삽입
10         if (t.getRight() != null) // 제거된 오른쪽 자식이 null이 아니면
11             q.add(t.getRight()); // 큐에 오른쪽 자식 삽입
12     }
13 }
```

- 큐 자료구조를 활용

- Line 02: 자바 라이브러리의 LinkedList를 사용해 구현한 Queue 사용
 - Queue 인터페이스를 연결 리스트로 구현한 것이 LinkedList (LinkedList implements Queue)
- Line 03: q 에서 삭제된 노드를 참조하기 위해 Node 타입의 지역변수를 선언

본인이 구현한 큐를 사용해도 됩니다.

이진트리 : 레벨순회

```
01 public void levelorder(Node root) { // 레벨순회
02     Queue<Node> q = new LinkedList<Node>(); // 큐 자료구조 이용
03     Node t;
04     q.add(root); // 루트 노드 큐에 삽입
05     while (!q.isEmpty()) {
06         t = q.remove(); // 큐에서 가장 앞에 있는 노드 제거
07         System.out.print(t.getKey()+" "); // 제거된 노드 출력(방문)
08         if (t.getLeft() != null) // 제거된 왼쪽 자식이 null이 아니면
09             q.add(t.getLeft()); // 큐에 왼쪽 자식 삽입
10         if (t.getRight() != null) // 제거된 오른쪽 자식이 null이 아니면
11             q.add(t.getRight()); // 큐에 오른쪽 자식 삽입
12     }
13 }
```

- Line 05의 while-루프: line 06에서 q의 가장 앞에 있는 노드를 삭제하고, 삭제한 노드의 레퍼런스를 t에 저장
 - Line 07: 큐에서 삭제된 노드를 방문
 - Line 08~11: t의 왼쪽 자식과 오른쪽 자식을 큐에 차례로 추가
 - 자식이 null인 경우, 큐에 추가하지 않음

이진트리 : 기타 연산

- `size()`: 트리의 노드 수 계산
- `height()`: 트리의 높이 계산
- `isEqual()`: 2개의 이진트리에 대한 동일성 검사

`size()`와 `height()`는 후위 순회 이용, `isEqual()`은 전위 순회 이용

이진트리 : 노드 수, 높이, 비교

- 트리의 **노드 수**를 계산하는 것은 트리의 아래에서 위로 각 자식의 후손노드 수를 합하며 올라가는 과정을 통해 수행되며, 최종적으로 루트노드에서 총 합을 구함
- 트리의 **높이**도 아래에서 위로 두 자식을 각각 루트노드로 하는 서브트리의 높이를 비교하여 보다 큰 높이에 1을 더하는 것으로 자신의 높이를 계산하며, 최종적으로 루트노드의 높이가 트리의 높이가 됨
- 2개의 이진트리를 **비교**하는 것은 다른 부분을 발견하는 즉시 비교 연산을 멈추기 위해 전위순회 방법을 사용

이진트리 : 노드 수

[핵심 아이디어]

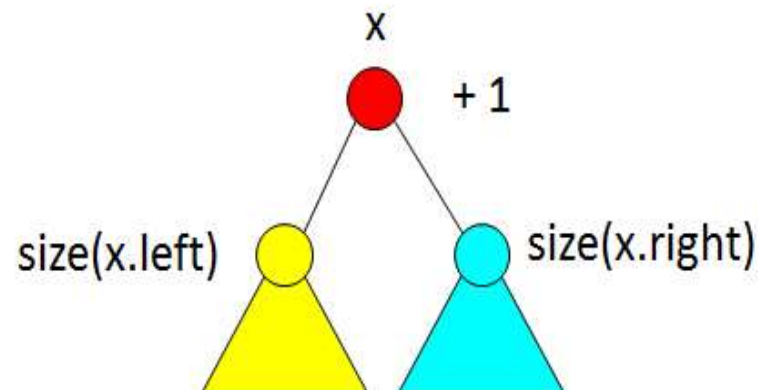
트리의 노드 수 = 1 +

(루트노드의 왼쪽 서브트리에 있는 노드 수) +

(루트노드의 오른쪽 서브트리에 있는 노드 수)

참고: +1은 루트노드 자신을 계산에 반영하는 것

왼쪽과 오른쪽 서브트리의 높이도 동일한 방식으로 계산



이진트리 : 노드 수

```
01 public int size(Node n) { // n를 루트로하는 (서브)트리에 있는 노드 수
02     if (n == null)
03         return 0; // null이면 0 리턴
04     else
05         return (1 + size( n.getLeft() ) + size( n.getRight() ));
06 }
```

- 루트노드를 인자로 전달하여 호출
- Line 02: 노드가 null이면 line 03에서 0을 리턴
- Null이 아니면 line 05에서 왼쪽 자식노드를 루트노드로 하는 서브트리의 노드 수와 오른쪽 자식노드를 루트노드로 하는 서브트리의 노드 수를 더한 결과에 1을 더한 값을 리턴

이진트리 : 트리의 높이

[핵심 아이디어]

트리의 높이 = 1 +

max (루트의 왼쪽 서브트리의 높이,

루트의 오른쪽 서브트리의 높이)

참고: +1은 루트노드 자신을 계산에 반영 (자식으로부터 부모까지 거리=1)

왼쪽과 오른쪽 서브트리의 높이도 동일한 방식으로 계산



이진트리 : 트리의 높이

```
01 public int height(Node n) { // n를 루트로하는 (서브)트리의 높이
02     if (n == null)
03         return 0; // null이면 0 리턴
04     else
05         return (1 + Math.max(height(n.getLeft()), height(n.getRight())));
06 }
```

- 루트노드를 인자로 전달하여 호출
- Line 02: 노드가 null이면, line 03에서 0을 리턴
- Null이 아니면 line 05에서 왼쪽 자식노드를 루트노드로 하는 서브트리 높이와 오른쪽 자식노드를 루트노드로 하는 서브트리의 높이 중에서 보다 큰 높이에 1을 더한 값을 리턴

이진트리 : 두개의 트리 비교 (같은가?)

[핵심 아이디어] 전위순회 과정에서 다른 점이 발견되는 순간 false를 리턴

- 두개의 비교하는 노드에 저장된 값이 다른 것을 발견하는 즉시 비교 연산을 멈추기 위해 전위 순위 방법을 사용함 (전위순회는 노드를 먼저 방문한다)

이진트리 : 두개의 트리 비교 (같은가?)

```
01 public static boolean isEqual(Node n, Node m){ // 두 트리의 동일성 검사
02     if(n==null || m==null) // 둘중에 하나라도 null이면
03         return n == m; // 둘다 null이면 true, 아니면 false
04
05     if (n.getKey().compareTo(m.getKey()) != 0) // 둘다 null이 아니면 item 비교
06         return false;
07
08     return( isEqual(n.getLeft(), m.getLeft()) && // item이 같으면 왼쪽 자식 재귀호출
09             isEqual(n.getRight(), m.getRight())); // 오른쪽 자식 재귀호출
10 }
```

- 비교하려는 두 트리의 루트노드를 인자로 전달하여 호출
- Line 02: 노드 n과 m 둘 중에 적어도 하나가 null인 경우
 - 만일 둘 다 null이면 true, 한 쪽만 null이면 트리가 다른 것이므로 false를 리턴
- Line 05 (둘 다 null이 아닌 상태): 두 노드의 키를 비교하여 다르면 (compareTo의 리턴 값이 0이 아니면) false 리턴 // 노드의 키를 먼저 확인하는 전위순회 기법
- 0이면 같은 key값을 갖는 경우이므로 line 08~09에서 각 트리의 왼쪽 자식노드와 오른쪽 자식노드를 인자로 하여 isEqual() 메소드를 재귀호출

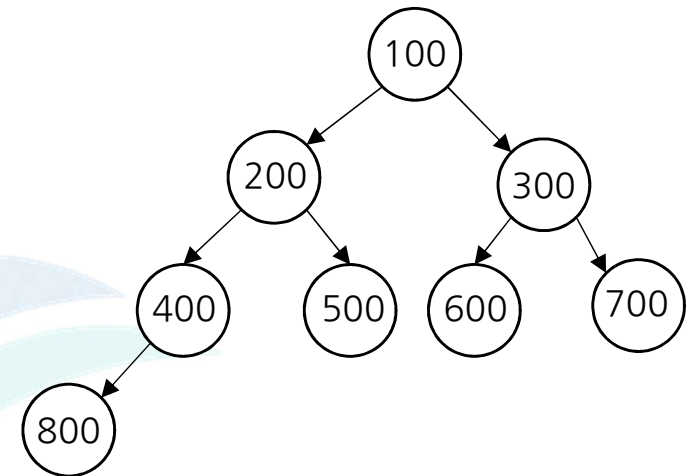
이진트리 : 수행시간

- 앞서 설명된 각 연산은 트리의 각 노드를 한 번씩만 방문하므로 $O(N)$ 시간 소요



이진트리 : 테스트

- main 메소드 구성 // 코드는 직접 작성하세요! (과제 1번)
 1. 다음과 같이 t1, t2 두개의 트리 구성
 2. t1 트리에서 size 메소드 호출
 3. t1 트리에서 height 메소드 호출
 4. t1 트리에서 preorder 메소드 호출
 5. t1 트리에서 inorder 메소드 호출
 6. t1 트리에서 postorder 메소드 호출
 7. t1 트리에서 levelorder 메소드 호출
 8. t1 과 t2 트리 비교하는 isEqual 메소드 호출

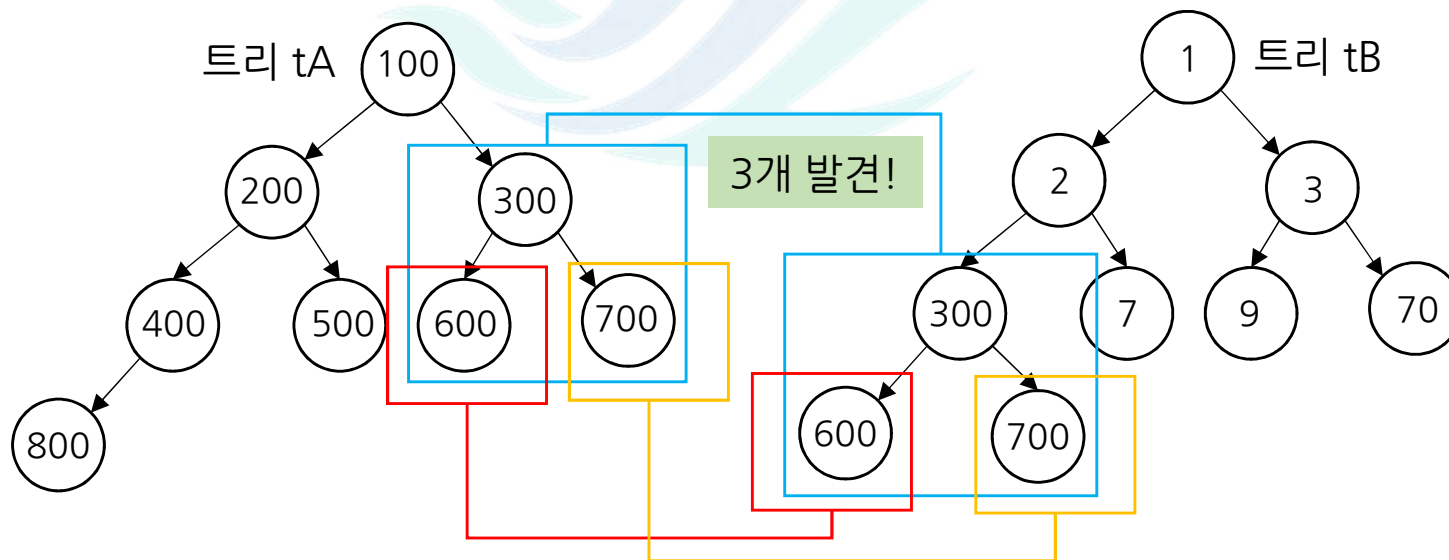


결과:

```
트리 노드 수 = 8
트리 높이 = 4
전위순회: 100 200 400 800 500 300 600 700
중위순회: 800 400 200 500 100 600 300 700
후위순회: 800 400 500 200 600 700 300 100
레벨순회: 100 200 300 400 500 600 700 800
동일성 검사: true
```

실습 과제

- 강의노트의 코드를 그대로 타이핑 하여 이진트리 구현 + 테스트를 위한 main 메소드 구현 및 실행
- 정수가 저장된 트리를 배열로 출력/저장하는 메소드와, 정수 배열을 입력으로 받아서 트리를 다시 복원하는 메소드 구현 (참고: p. 23, p. 24)
- 두개의 트리에서 동일한 서브 트리가 있는지 검사하는 메소드 (있다면 해당하는 모든 서브 트리를 레벨 순회로 출력)



끝.

질문?

