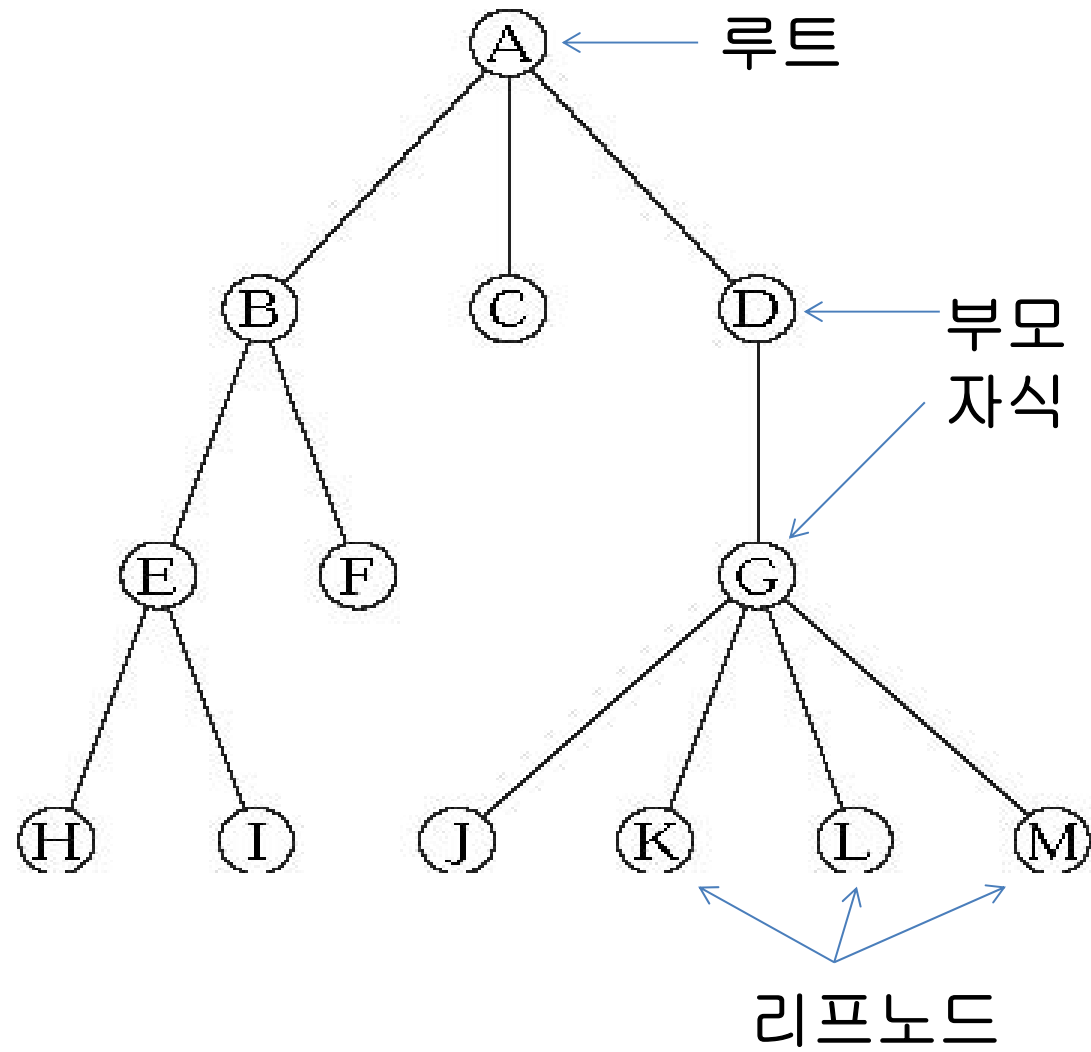
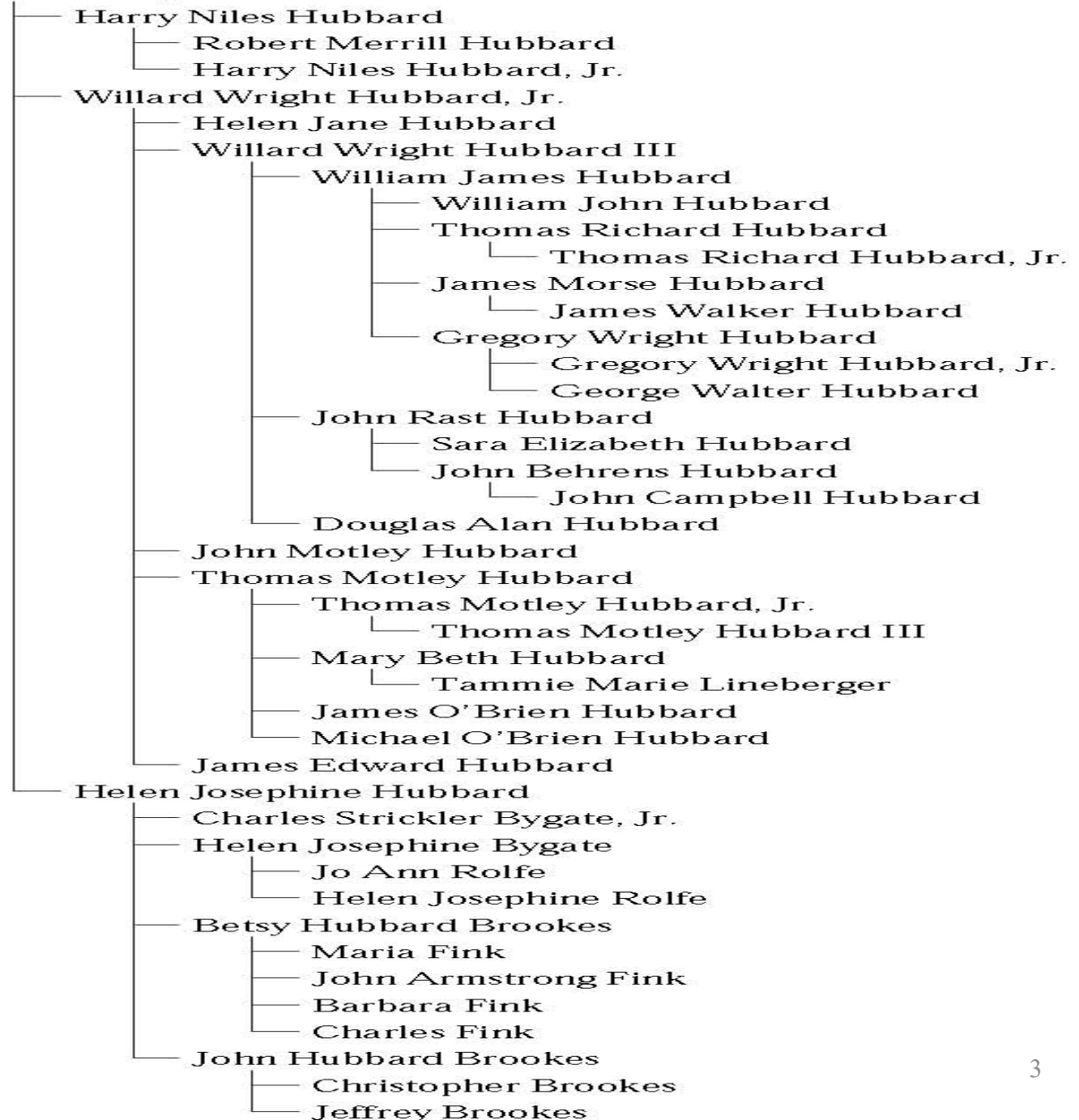


11. 트리

추상 트리



Willard Wright Hubbard



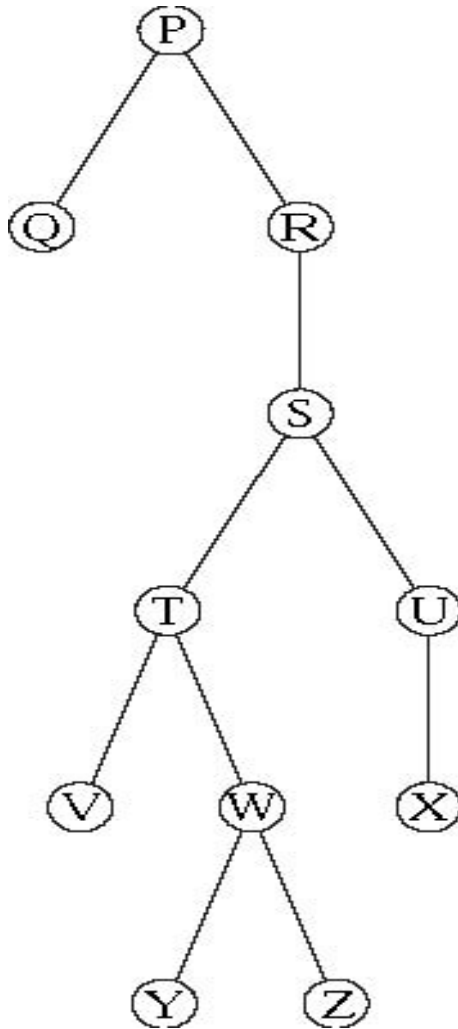
11.1 트리

- 경로 : 인접 노드의 시이퀀스
 - 루트 경로 : 루트로부터 해당 노드까지의 유일한 경로
 - 루트-리프 경로 : 리프 노드에 이르는 루트 경로
 - 경로의 길이 : 부모-자식 쌍의 개수. 노드의 개수보다는 하나 적음
- 트리의 크기 : 노드의 개수
 - 공백 트리 : 크기가 0인 유일한 트리
 - 단독 트리 : 크기가 1인 트리
- 서브트리와 슈퍼트리
 - T_1 의 모든 노드가 T_2 의 노드이고, T_1 에 있는 y 의 부모인 x 가 동시에 T_2 에 있는 y 의 부모일 때, T_1 은 T_2 의 서브트리가 됨
 - T_2 은 T_1 의 슈퍼트리라고 함

11.2 트리의 특성 (1)

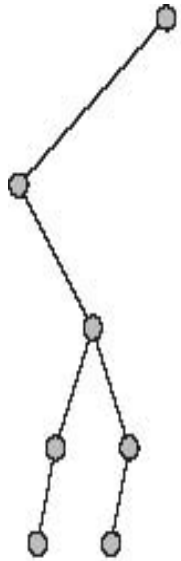
- 트리의 높이(height)
 - 최장 루트-리프 경로¹의 길이와 같음
- 노드의 깊이(depth)
 - 노드의 루트 경로의 길이
- 레벨
 - 주어진 깊이에 있는 모든 노드로 구성됨
- 트리의 경로 길이
 - 트리에 있는 모든 노드에 대한 깊이의 합
- 트리의 너비
 - 최대 레벨 크기
- 노드의 차수²
 - 자식들의 개수
- 트리의 차수
 - 노드들의 차수 중에서 최대 차수

경로 길이가 32인 트리



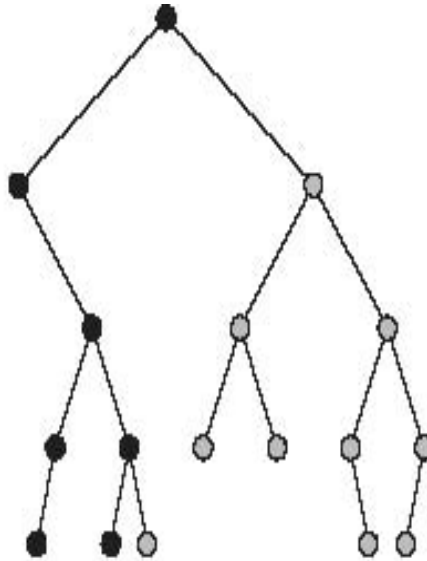
리프노드 : 자식이 없는 노드
리프노드가 아닌 노드들을 내부노드 라고함
높이 : 6
노드(크기)는 11개, 링크는 10개, 노드-1은 링크의 수

차수가 ?이고 높이가 ?인 트리



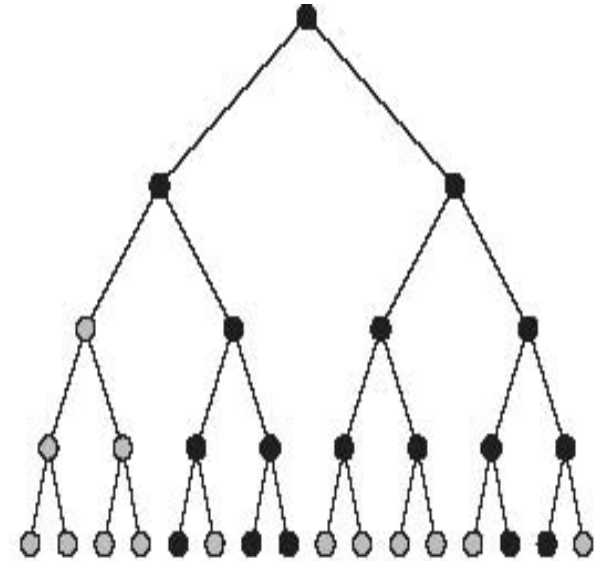
$d = \square$ $h = \square$ $n = \square$
차수 높이 크기

2, 4, 7



$d = \square$ $h = \square$ $n = \square$

2, 4, 17



$d = \square$ $h = \square$ $n = \square$

2, 4, 31
full 트리

트리의 특성 (2)

- 포화 트리 (full tree)
 - 모든 리프가 같은 레벨에 있고, 모든 노드가 동일한 차수를 가지고 있음
 - 각 노드의 차수는 트리의 차수와 같음
 - 각각의 차수와 높이를 가지는 포화 트리는 유일함
- 포화 트리의 크기

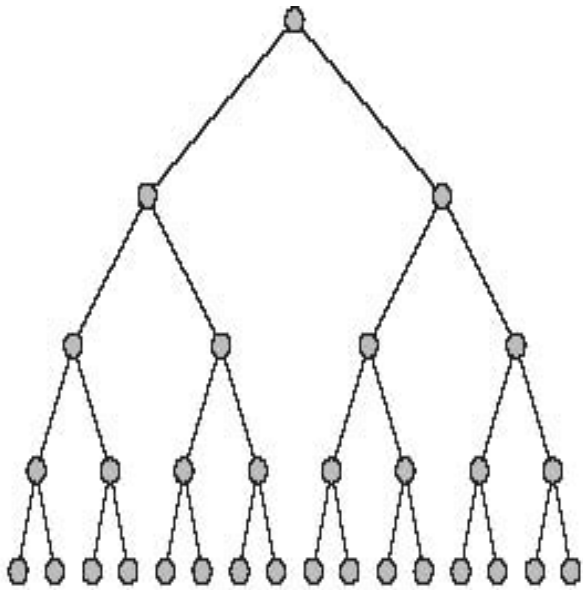
$$n = \frac{d^{h+1} - 1}{d - 1}$$

- 포화트리의 높이 $h = \theta(\log n)$

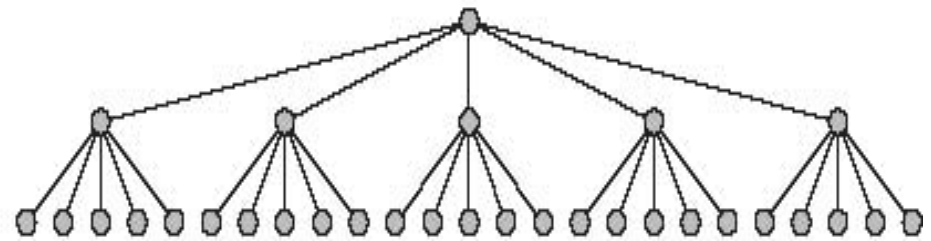
- 높이가 h이고 차수가 d인 트리의 크기 n에 대한 범위

$$h + d \leq n \leq \frac{d^{h+1} - 1}{d - 1}$$

포화(full) 트리



$$d = 2, h = 4, w = 16, n = 31$$

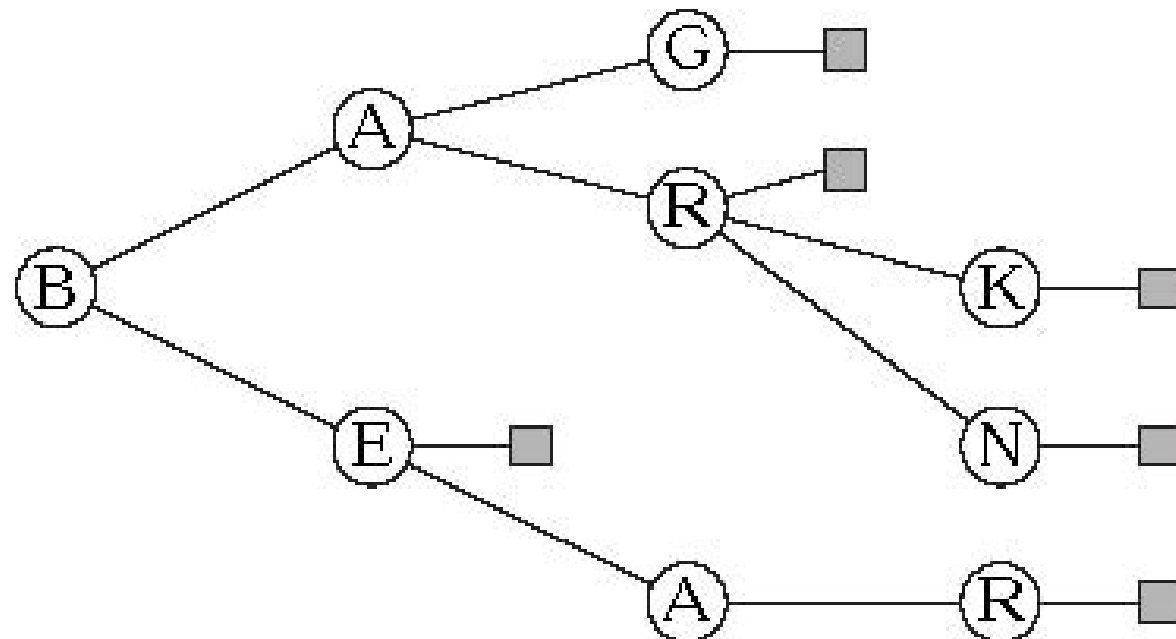


$$d = 5, h = 2, w = 25, n = 31$$

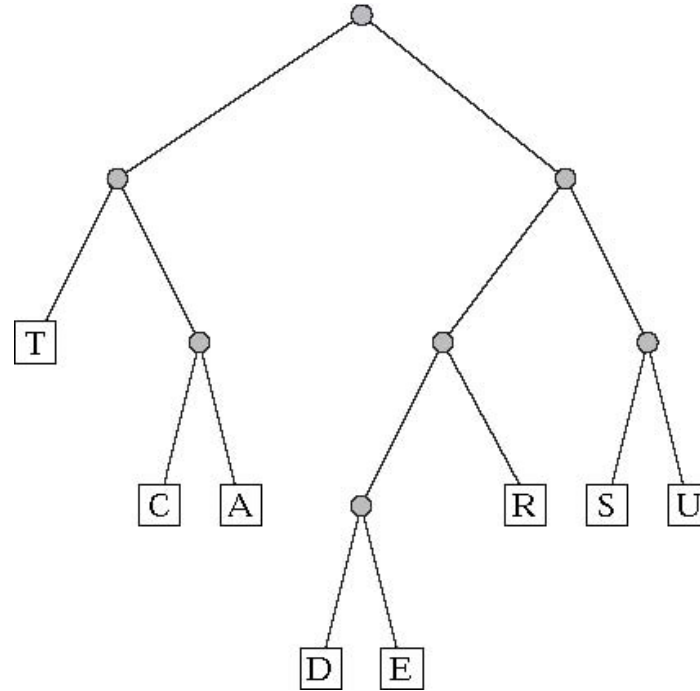
트리의 특성 (3)

- 내부 노드(internal node)
 - 리프가 아닌 노드
- 외부 노드(external nodea)
 - 리프 노드가 내부노드와 다른 데이터 타입을 사용하여 구현되거나 데이터가 없는 더미 노드로 사용되는 등 리프노드가 특수한 목적으로 사용되는 트리에서 리프노드를 외부 노드라고 부름
- 내부 경로 길이
 - 모든 내부 노드의 경로 길이 합
- 외부 경로 길이
 - 어떤 레벨에서 외부 노드의 개수와 레벨을 곱한 값으로 주어지는 가중치의 합계

철자 트리



허프만 코드 트리

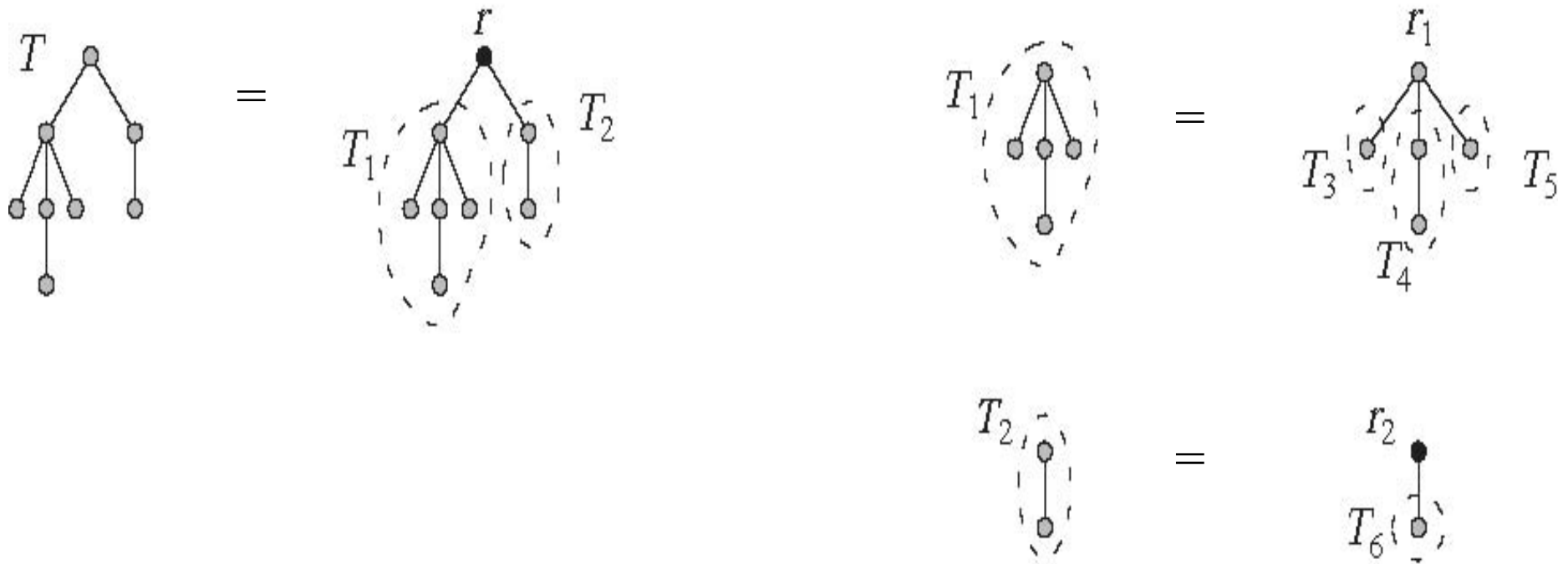


내부 경로 길이 : 11
외부 경로 길이 : 25

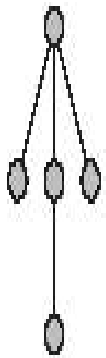
11.3 트리의 순환 정의

- 정의
 - 트리는 공집합이거나, r 이 노드이고 S 가 r 을 포함하지 않고 서로 분리된 트리의 집합일 때 쌍 (r, S) 이다.
 - 노드 r 을 트리의 루트라고 부르고 S 에 있는 트리를 서브트리 (subtree)라고 부른다.

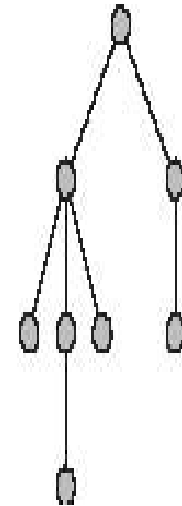
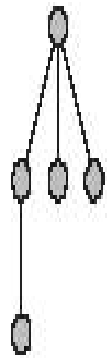
트리의 순환 정의를 검증



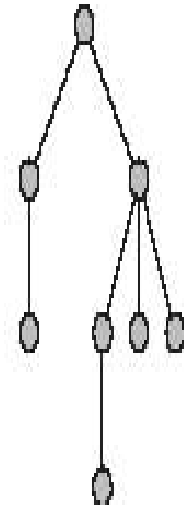
무순서 트리



=



=

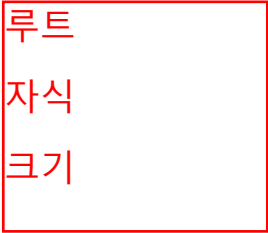


무순서 트리 클래스

- LISTING 11.1: An UnorderedTree Class

```
1 class UnorderedTree {
2     private Object root;
3     private Set subtrees;
4     private int size;

5     public UnorderedTree() { // constructs the empty tree
6     }
7
8     public UnorderedTree(Object root) { // constructs a singleton
9         this.root = root;
10        subtrees = new TreeSet(); // constructs the empty set
11        size = 1;
12    }
```




```
14  public UnorderedTree(Object root, Set trees) {
15      this(root);
16      for (Iterator it=trees.iterator(); it.hasNext(); ) {
17          Object object=it.next();
18          if (object instanceof UnorderedTree) {
19              UnorderedTree tree = (UnorderedTree)object;
20              subtrees.add(tree);
21              size += tree.size();
22          }
23      }
24  }
26  public int size() {
27      return size;
28  }
29 }
```

다음의 트리를 구축해 보자

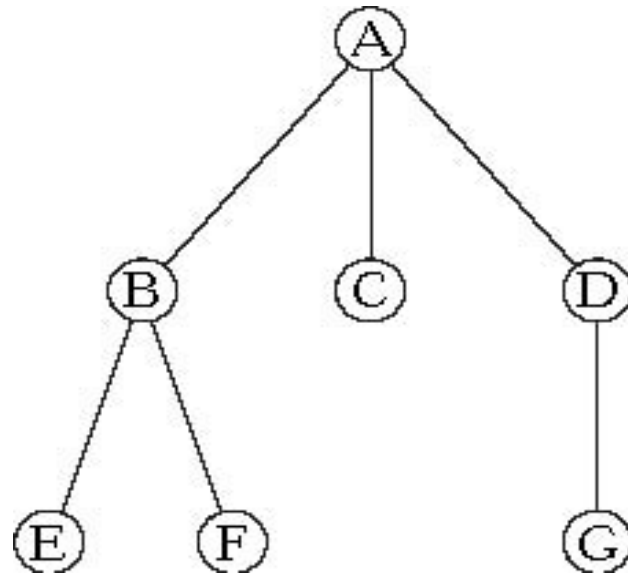


그림 11.3

무순서 트리 구축

- `UnorderedTree treeA, treeB, treeD;`
`UnorderedTree treeC = new UnorderedTree("C");`
`UnorderedTree treeE = new UnorderedTree("E");`
`UnorderedTree treeF = new UnorderedTree("F");`
`UnorderedTree treeG = new UnorderedTree("G");`
- `// build subtree rooted at B:`
`Set subtreesOfB = new Set();`
`subtreesOfB.add(treeE);`
`subtreesOfB.add(treeF);`
`treeB = new UnorderedTree("B", subtreesOfB);`

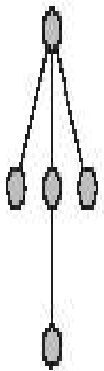
무순서 트리 구축

- `// build subtree rooted at D:
Set subtreesOfD = new Set();
subtreesOfD.add(treeG);
treeD = new UnorderedTree("D", subtreesOfD);`
- `// build subtree rooted at A:
Set subtreesOfA = new Set();
subtreesOfA.add(treeB);
subtreesOfA.add(treeC);
subtreesOfA.add(treeD);
treeA = new UnorderedTree("A", subtreesOfA);`

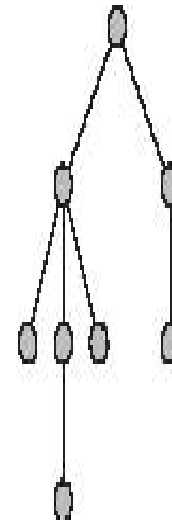
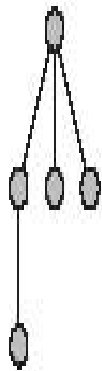
11.5 순서 트리

- 정의
 - 순서 트리(ordered tree)는 공집합이거나, r 이 노드이고 S 가 r 을 포함하지 않고 서로 분리된 트리의 **시이퀀스**일 때 쌍 (r, S) 이다.

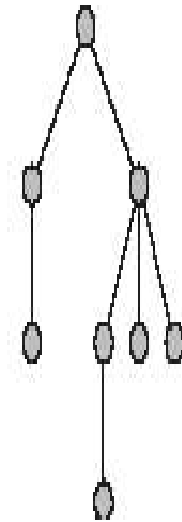
동일하지 않은 순서 트리



\neq



\neq



```
import java.util.*;
```

```
public class OrderedTree {
```

```
    private Object root;
```

```
    private List subtrees;
```

```
    private int size;
```

```
    public OrderedTree() { // constructs the empty tree  
    }
```

```
    public OrderedTree(Object root) { // constructs a singleton  
        this.root = root;  
        subtrees = new LinkedList(); // constructs the empty list  
        size = 1;
```

```
    }
```

```
    public OrderedTree(Object root, List trees) {
```



```
    }
```

```
    public int size() {  
        return size;
```

```
    }
```

```
}
```

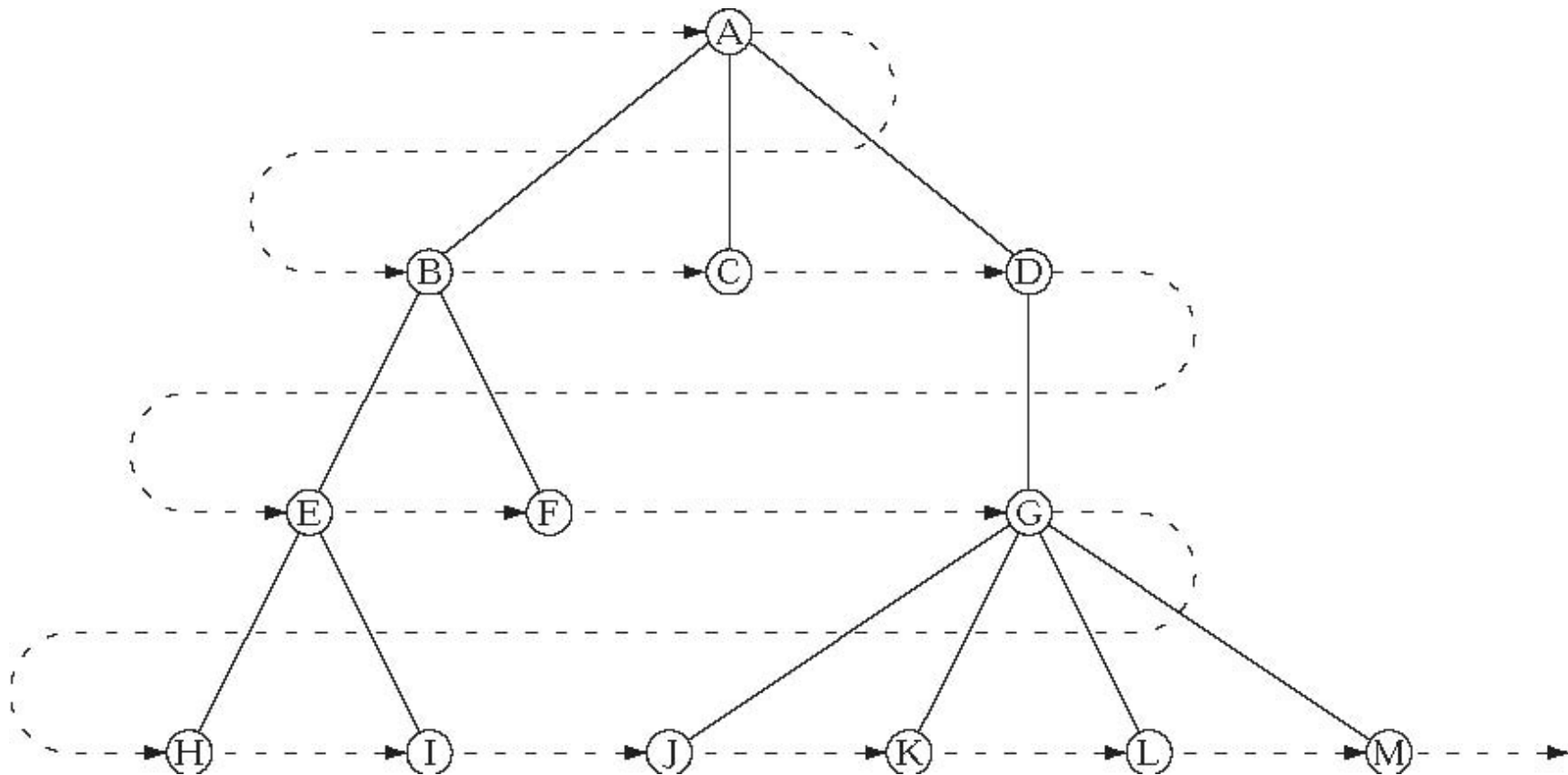
프로그래밍문제 11.1

OrderedTree 클래스 구현







11.6 순서 트리를 위한 순회 알고리즘

- 레벨 순서(level order) 순회
 - 트리의 레벨에 따른 순서로 이동
- 전위(preorder) 순회 (루트 노드 우선)
 1. 루트를 방문한다.
 2. 각 서브트리를 순서대로 전위 순회한다.
- 후위(postorder) 순회 (루트 노드 나중)
 1. 각 서브트리를 순서대로 후위 순회한다.
 2. 루트를 방문한다.

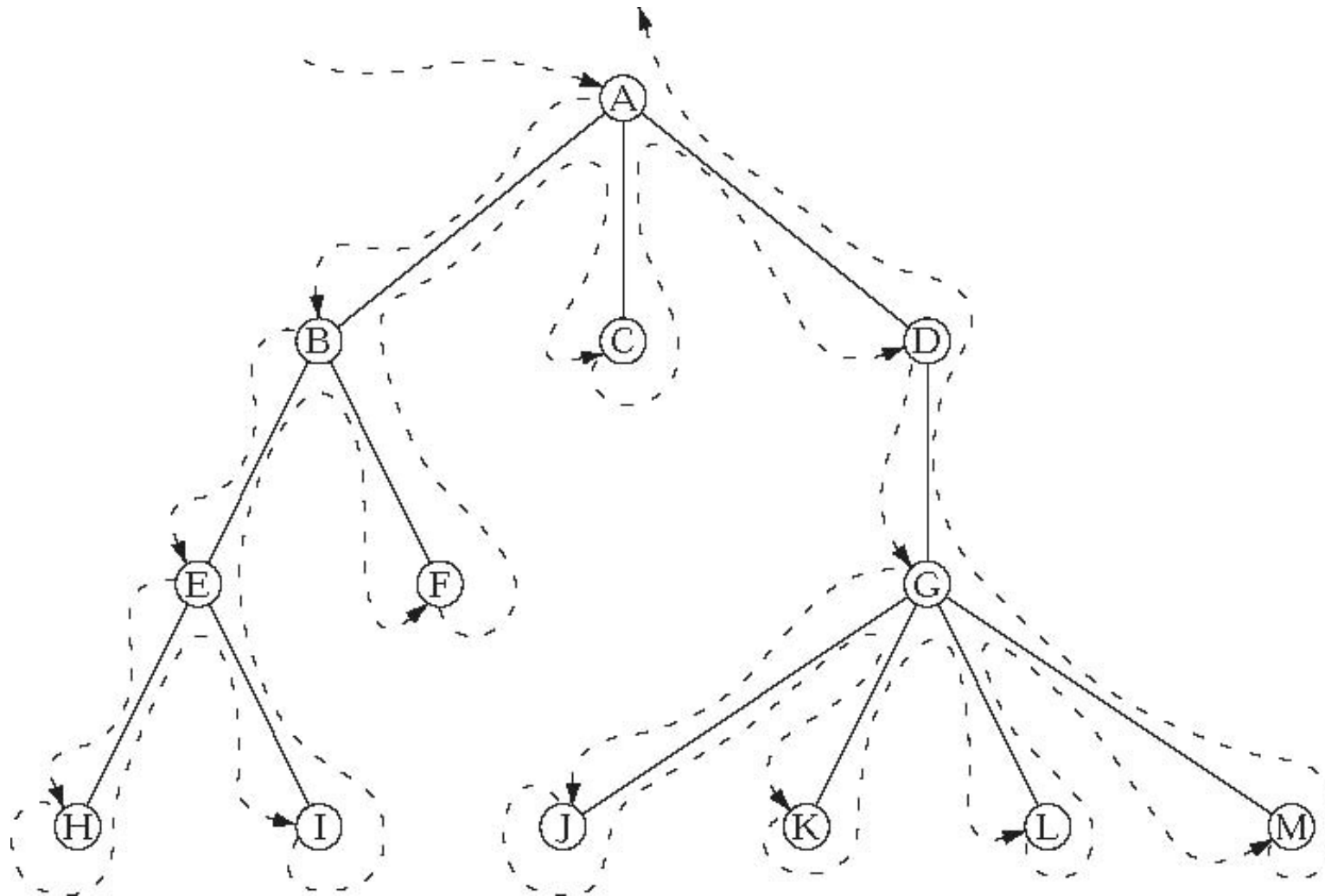
레벨 순서 순회



레벨 순서(level order) 순회

1. 를 초기화한다. 
2. 루트를 에 삽입한다.
3. 가 공백이 될 때까지 단계 4-6을 반복한다.
4. 에서 첫 번째 노드 x 를 삭제한다.
5. x 를 방문한다.
6. x 의 모든 자식들을 순서대로 에 삽입한다.

전위 순회



OrderedTree 클래스에 다음의 메소드 추가

- ```
public void preorderPrint(){
 //루트를 방문한다.
 //각 서브트리를 순서대로 전위 순회한다.
}
```
- ```
public void postorderPrint( ){  
    //각 서브트리를 순서대로 후위 순회한다.  
    //루트를 방문한다.  
}
```

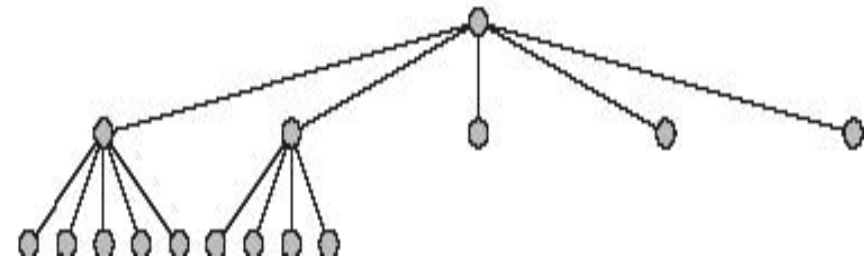
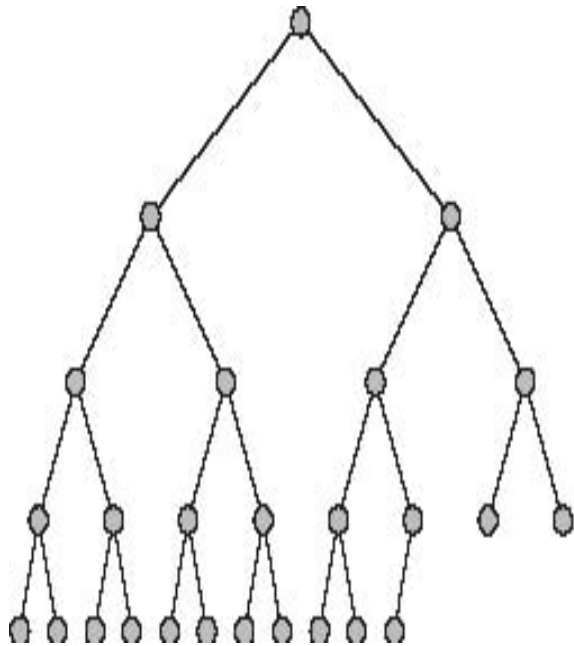
11.7 완전 순서 트리

- 선형화 또는 직렬화
 - 트리를 메모리나 디스크에 저장하는 방법
 - 세 가지 순회 알고리즘 중 어떤 것도 트리를 선형화하는데 사용할 수 있음
 - 자연 매핑 : 레벨 순서 순회를 사용하여 선형화하는 과정
- 자연 인덱스 번호 또는 노드 인덱스
 - 트리의 레벨 순서 순회에 따라 순서적으로 매겨져 있는 번호

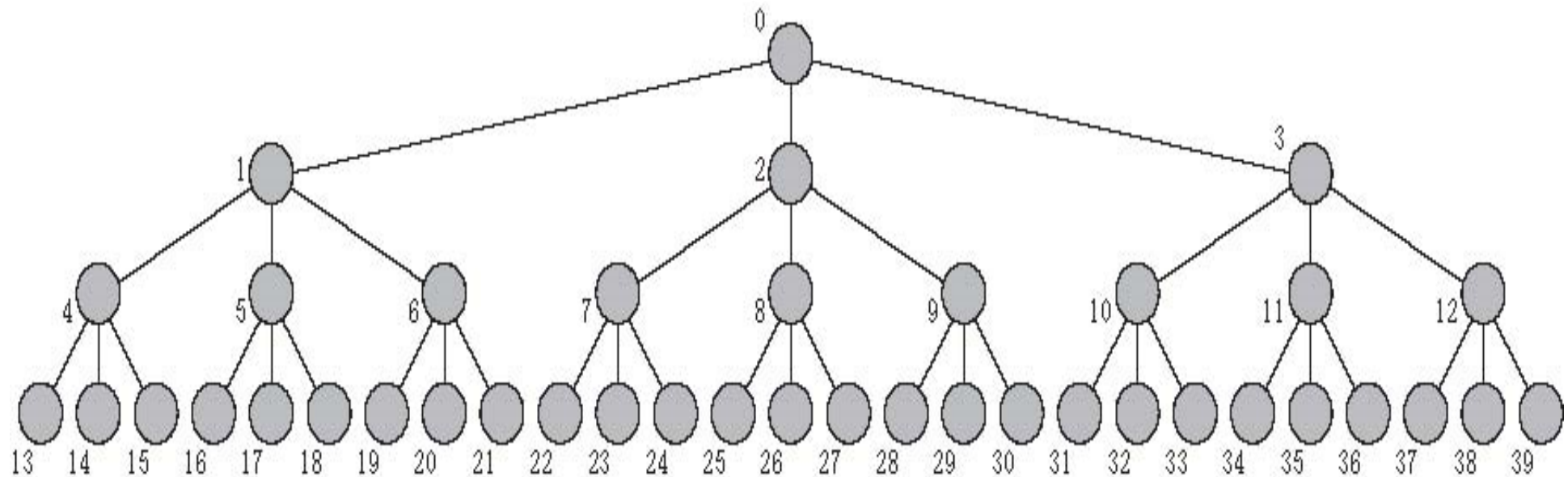
11.7 완전 순서 트리

- 완전 순서 트리(complete ordered tree)
 - 최하위 레벨의 노드 중 오른쪽 몇 개의 원소가 없는 것을 제외하고는 포화 트리와 같은 순서 트리
 - 이러한 트리는 미사용 원소가 없이 배열에 저장 가능하고, 배열의 크기는 트리에 있는 노드의 개수와 같음

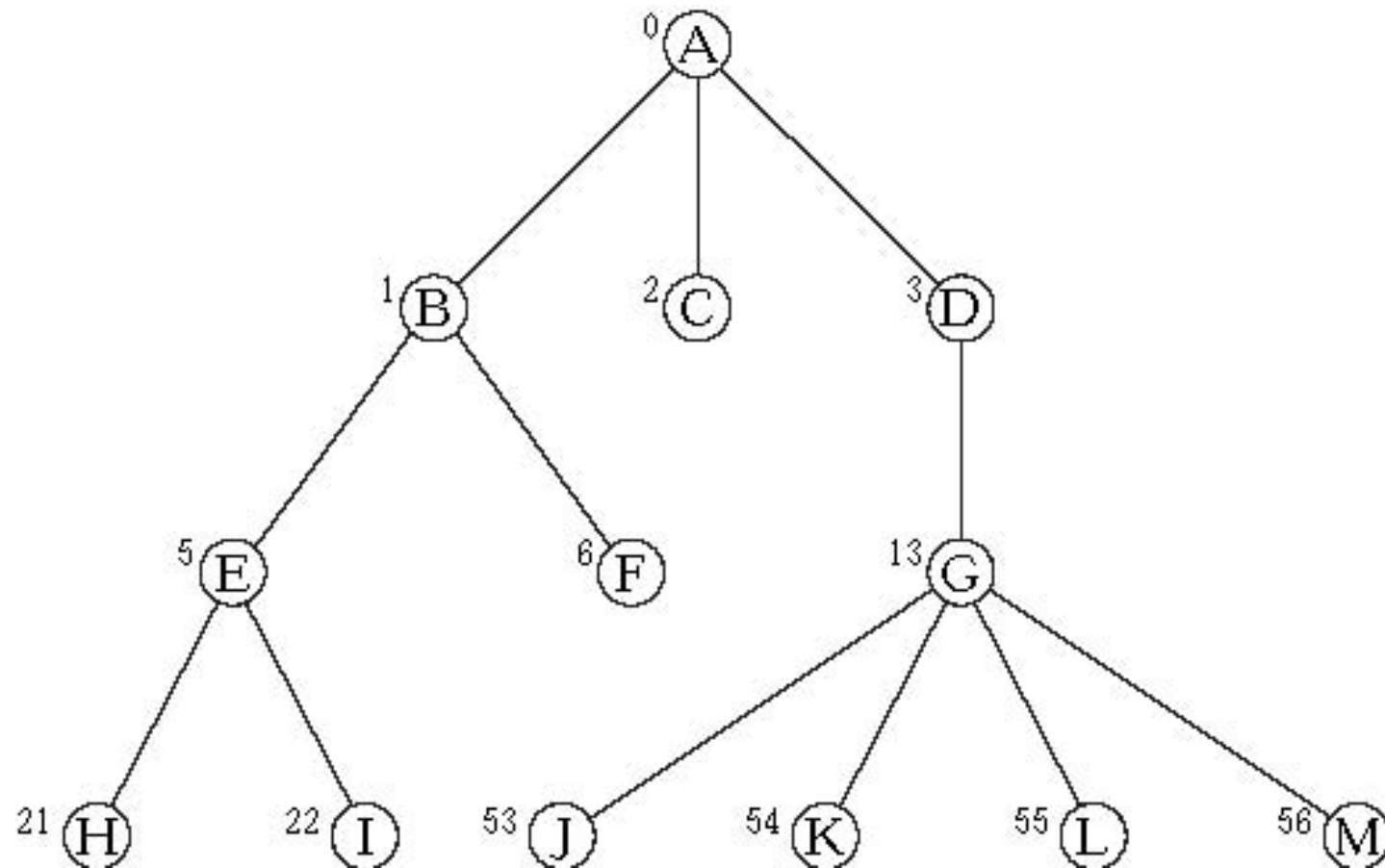
완전 순서 트리



차수가 3인 순서 트리의 직렬화



희소 순서 트리의 노드 인덱스 번호



- [공식 11.4] 순서트리의 레벨

차수가 d 인 포화순서트리의 레벨 i 에서 $k=(d^i - 1)/(d-1)$ 일 때, 노드의 자연 인덱스 번호는 $k, k+1, \dots, k+d^i - 1$ 이다

- [공식 11.5] 순서트리의 직렬화

차수가 d 인 포화순서트리에서, 노드 i 의 경우 부모의 인덱스는 $(i-1)/d$ 이 되고, 자식들은 $di+1, di+2, \dots, di+d$ 로 순서가 매겨진다.

(힌트: 먼저 노드 x 가 같은 레벨에 있는 그것의 왼쪽에 k 개의 노드를 가지고 있다면, x 의 자식들은 같은 레벨에 있는 그것들의 왼쪽에 반드시 dk 와 $dk+d-1$ 개 사이의 노드를 가지고 있어야 한다는 것을 보이시오.)