Nonlinear Data Structure
비선형자료구조

http://smartlead.hallym.ac.kr

Instructor: Jin Kim
010-6267-8189(033-248-2318)
jinkim@hallym.ac.kr
Office Hours:

Define a linear and non linear data structure.

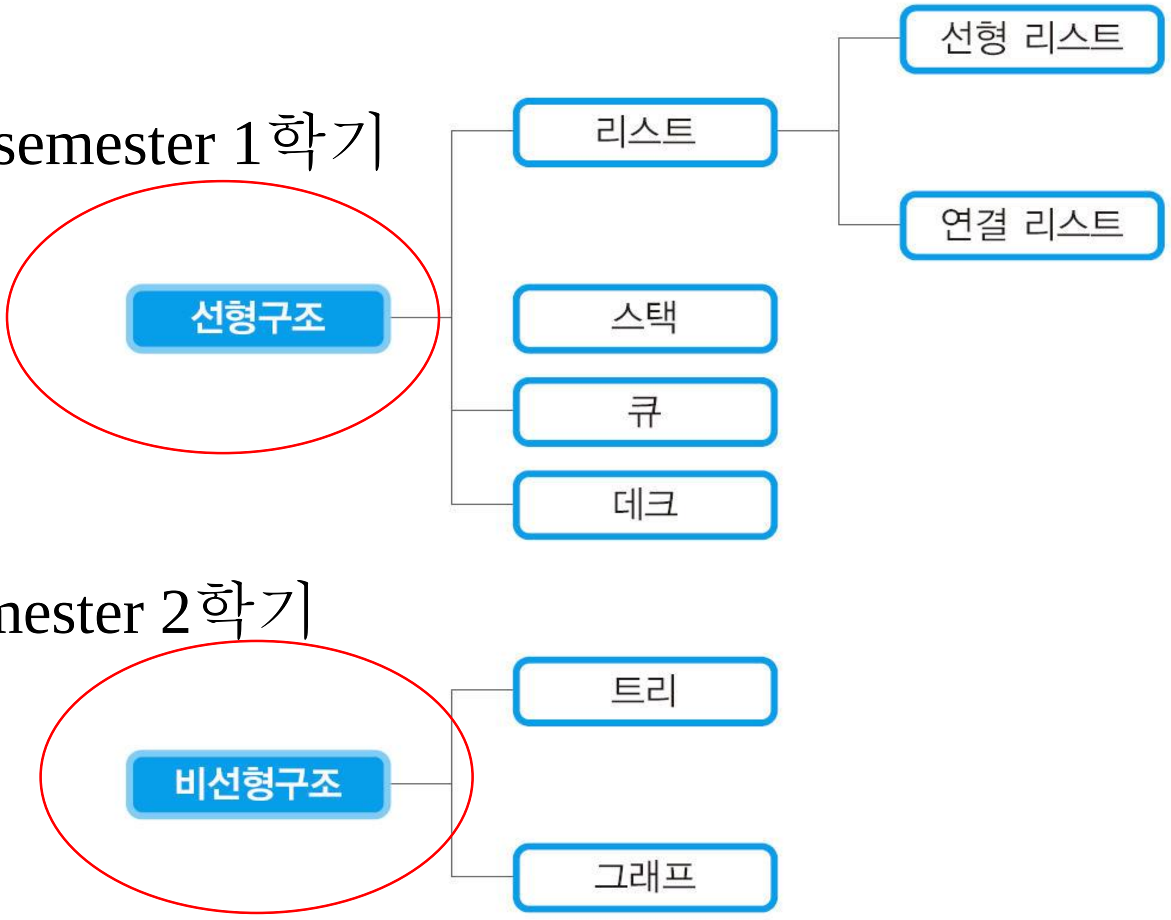
선형 비선형 자료구조

- **Linear data structure(선형자료구조):** a linear data structure traverses the data elements sequentially, in which only one data element can directly be reached. Ex: arrays, stack, queue 데이터가 일직선상에 위치
- **Non-linear data structure(비선형 자료구조):** every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure. Ex: trees, graphs 데이터가 선형으로 위치하지 않음

Linear and nonlinear data structure

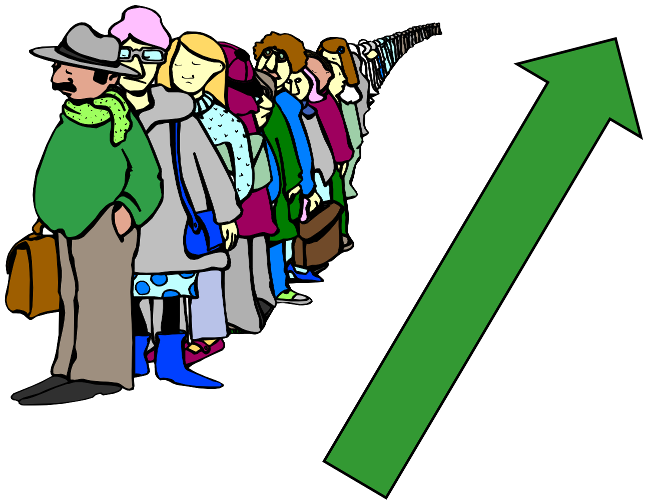
Spring semester 1학기

Fall semester 2학기



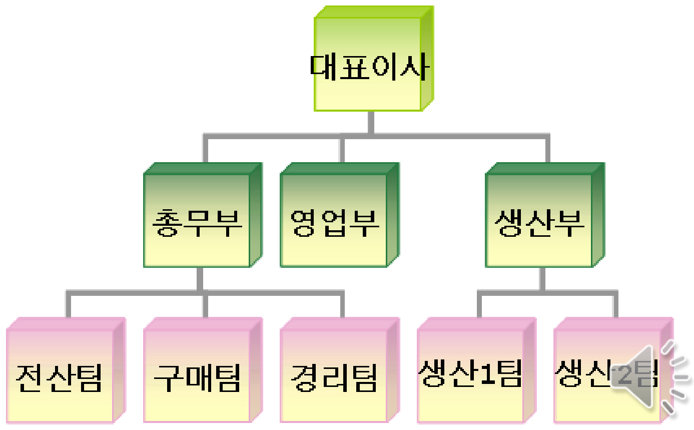
Linear(선형) Data Structures

- Elements have an order
- Each element has a predecessor(선행자) and a successor(후행자)
- 내앞사람은 선행자, 내 뒷사람은 후행자
- Examples:
  - List
  - Stack
  - Queue



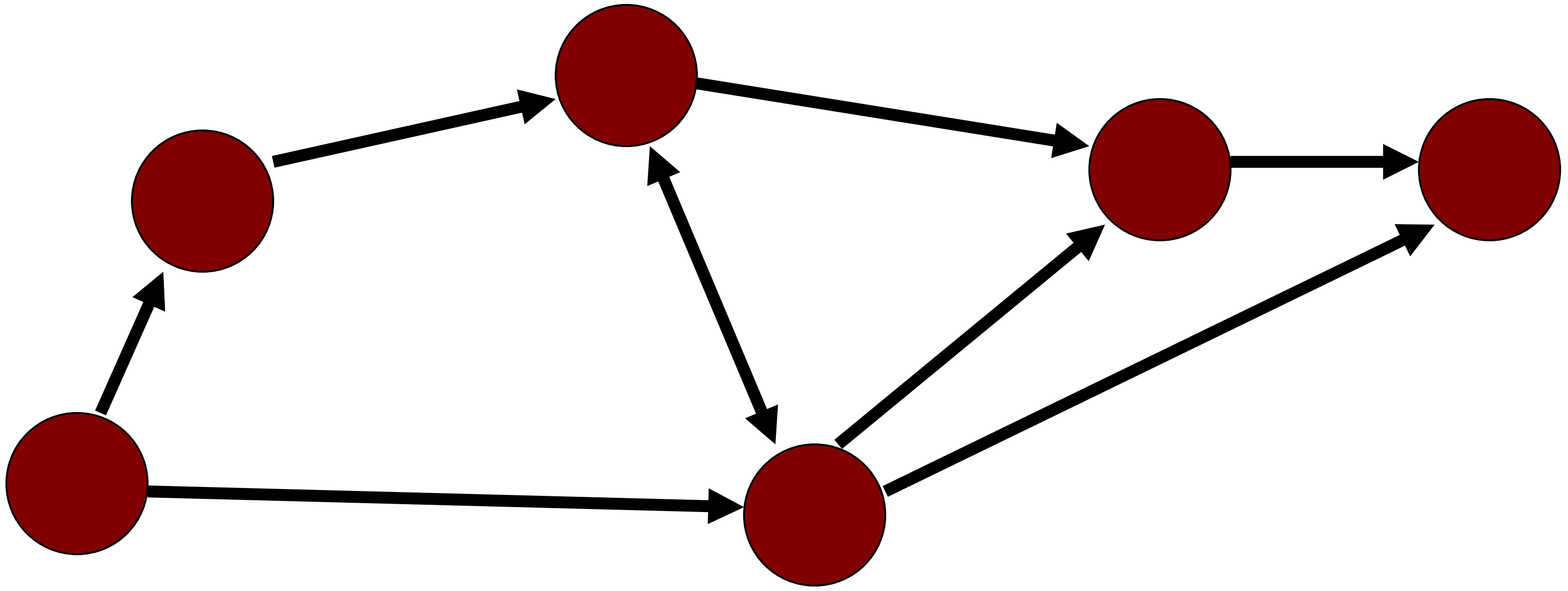
Hierarchical(nonlinear) Data Structures 계층구조적(비선형)

- Elements relate in 1:many relationships
  - Trees
- One **root(루트)** at the base of the structure
- One or more **leaves(리프)** most distant from the root
- 0, 1, or many **internal nodes내부노드** (neither root, nor leaf)
- Each node has:
  - a unique predecessor, and
  - 0, 1, or many successors



Graph(nonlinear) Data Structure 그래프(비선형)

- Elements may have many:many relationships with other elements원소들은 다대다관계
- No constraints on numbers of predecessors or successors 선행자와 후행자가 없음



Non Linear Data Structure

- Data structure we will consider this semester:
  - Tree
  - Binary Search Tree
  - Graph
  - Weighted Graph
  - Sorting
  - Balanced Search Tree

Trees: Outline

- Introduction(소개)
  - Representation Of Trees(트리를 표현하자)
- Binary Trees (이진트리)
- Binary Tree Traversals (이진 트리 순회)
- Additional Binary Tree Operations (이진트리 기타 연산)
- Threaded Binary Trees (스레드 이진 트리)
- Forests(숲, 나무들의 모임)
- General Trees to Binary Trees(일반트리에서 이진트리로)

Why a tree?왜 트리구조?

- Faster than linear data structures(선형자료구조보다 빠름)
- More natural fit for some kinds of data(어떤 종류의 데이터에는 트리를 사용하는 ㄴ것이 자연스러움)
  - Examples? Wll study(공부하자)

Introduction (1/8)

- **Definition** (**recursively**)재귀적정의: A *tree* is a finite set of one or more nodes such that
  - There is a specially designated node called ***root***.(루트)
  - The remaining nodes are partitioned into *n*>=0 disjoint set T_1,…, 쑤, where each of these sets is a tree. T_1,…, 쑤 are called the ***subtrees*** of the root.(서브트리)
- Every node in the tree is the root of some subtree(모든 노드들은 특정 서브트리의 루트)
- Recursive definition of tree
  - (재귀를 사용하여트리정의)
  - Tree= root + subtrees
  - 트리=루트+서브트리

Introduction (2/8)

- Some **Terminology(용어)**
  - *Root(루트)*: node without parent
  - *Node(노드)*: the item of information plus the branches to each node.
  - *Degree(차수) of a node*: the number of subtrees of a node
  - *degree* of a tree(트리차수): the maximum of the degree of the nodes in the tree.
  - *terminal* nodes단말노드 (or *leaf*리프): nodes that have degree zero
  - *nonterminal* nodes내부노드: nodes that don't belong to terminal nodes.
  - *Children*자식들: the roots of the subtrees of a node X are the *children* of X
  - *Parent*부모: X is the *parent* of its children.

Introduction (3/8)

- Some **Terminology** (용어 계속cont'd)
  - *Siblings동기, 형제, 자매*: children of the same parent are said to be siblings.
  - *Ancestors* of a node조상들: all the nodes along the path from the root to that node.(parent,grandparent,grand-grand parent, etc.)
  - *Descendants* of a node후손: all the nodes along the path from that node to the leaves.(child,grandchild,grand-grandchild)
  - The *level* of a node레벨: defined by letting the root be at level one. If a node is at level *l*, then it children are at level *l+1*.
  - *Height* (or *depth*)높이, 깊이: the maximum level of any node in the tree

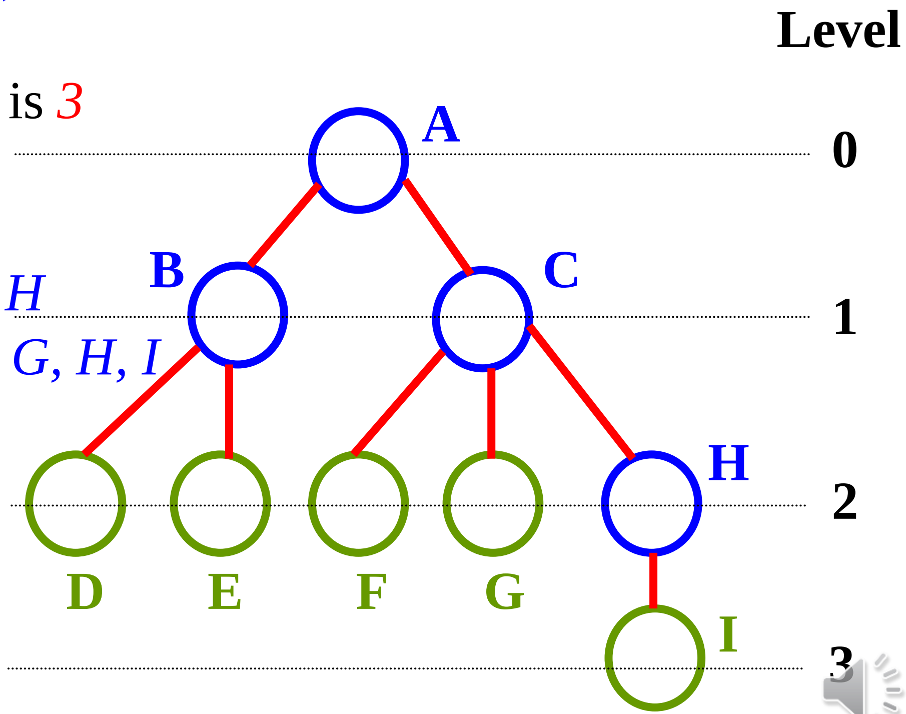
Introduction (4/8)

- Example
  - *A* is the *root 루트* node
  - *B* is the *parent 부모* of D and E
  - *C* is the *sibling 남매* of B
  - *D* and *E* are the *children 자식* of B
  - *D, E, F, G, I* are *external nodes*, or *leaves(외부노드, 혹은 나뭇잎)*
  - *A, B, C, H* are *internal nodes(내부노드)*
  - The *level 레벨* of *E* is *2*
  - The *height 높이(depth 깊이)* of the tree is *3*
  - The *degree 차수* of node *B* is *2*
  - The *degree* of the tree is *3*
  - The *ancestors 조상들* of node *I* is *A, C, H*
  - The *descendants 후손들* of node *C* is *F, G, H, I*

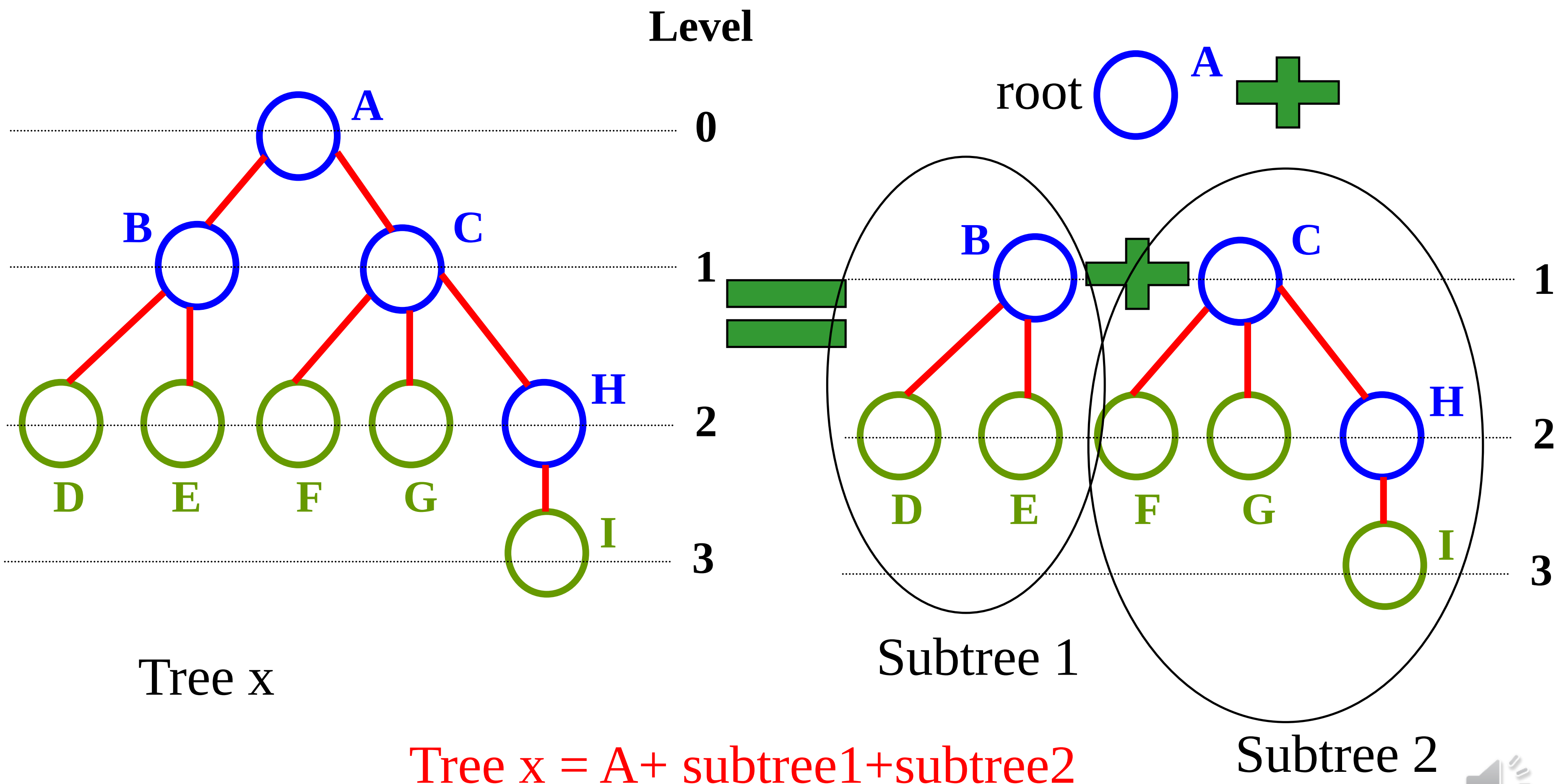
Property(특징): (# *edge*s) = (#*node*s) – 1

간선의수=정점의수 – 1

빨강선이 간선



교재에 따라 레벨이 1부터 시작하는 경우도 있다.
우리는 레벨이 0부터 시작한다고 약속하자.



Tree x = A+ subtree1+subtree2

Introduction (5/8)

- Representation Of Trees(트리표현)
  - List Representation(리스트로 표현)리스트는 원소를늘어놓은 것
    - we can write of Figure 5.2 as a list in which each of the subtrees is also a list

      (A (B (E (K, L), F), C (G), D (H (M), I, J)))
    - The root comes first,
      followed by a list of sub-trees

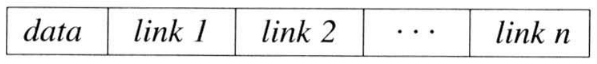


Figure 5.3: Possible list representation for trees

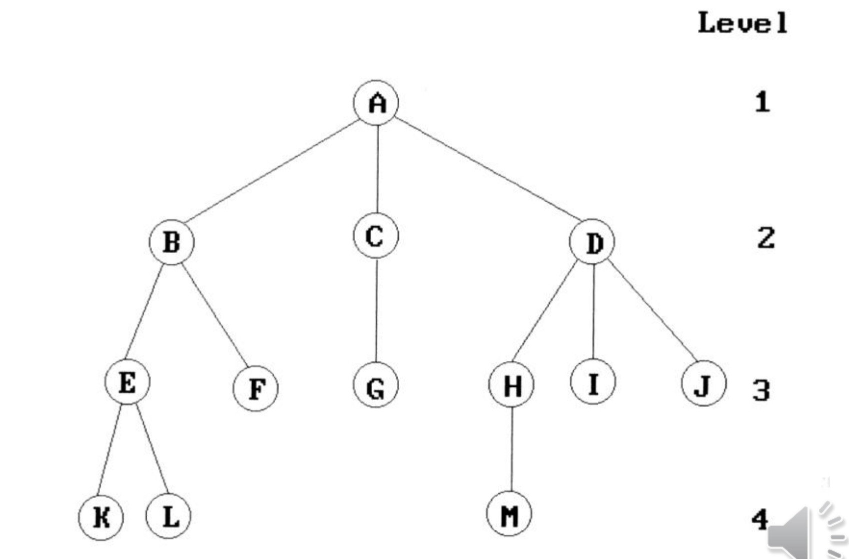


Figure 5.2: A sample tree

Introduction(6/8)

- Representation of tree? 트리표현?
- Obvious Pointer-Based Implementation: Node with value and pointers to children(연결리스트로 표현)
  - Problem문제? Different number of children (자식의 개수가 다르다)

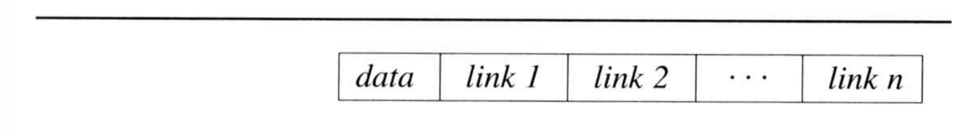
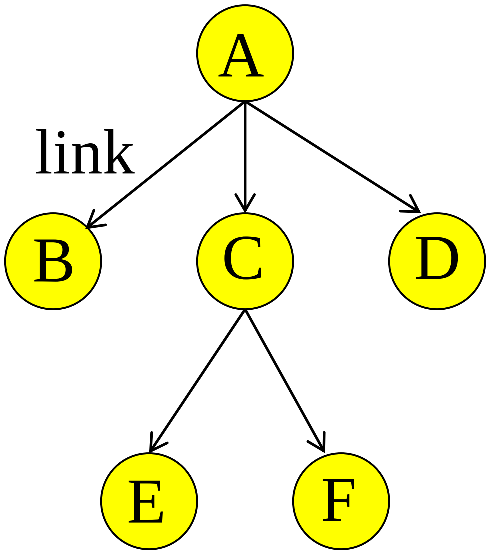


Figure 5.3: Possible list representation for trees

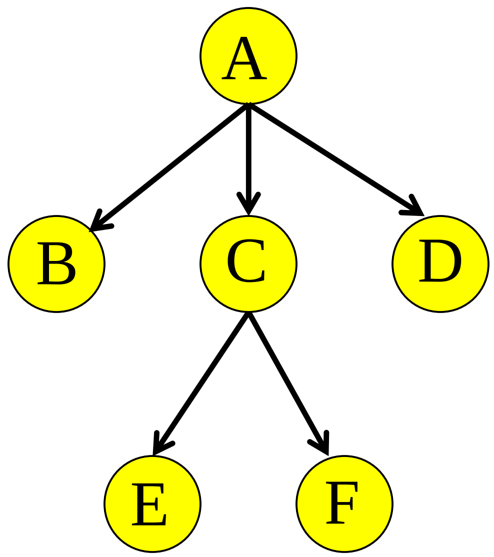
자식이 n 개면 링크도 n개

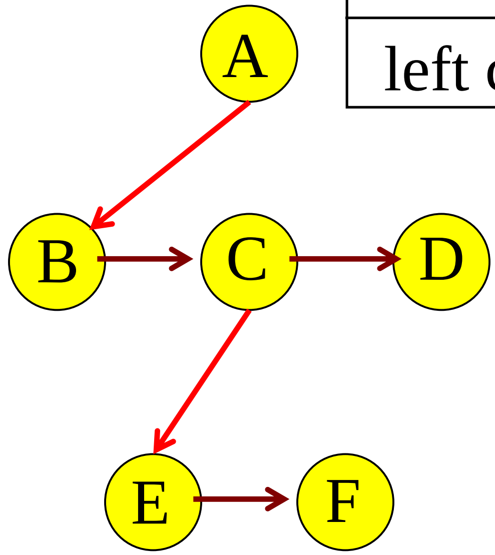


Introduction(7/8)

- Each node has 2 pointers각노드는 2개의 포인터를 가짐: one to its first child(첫째자식) and one to right sibling(오른쪽남매)
- Left Child-Right Sibling Representation왼쪽자식오른쪽남매

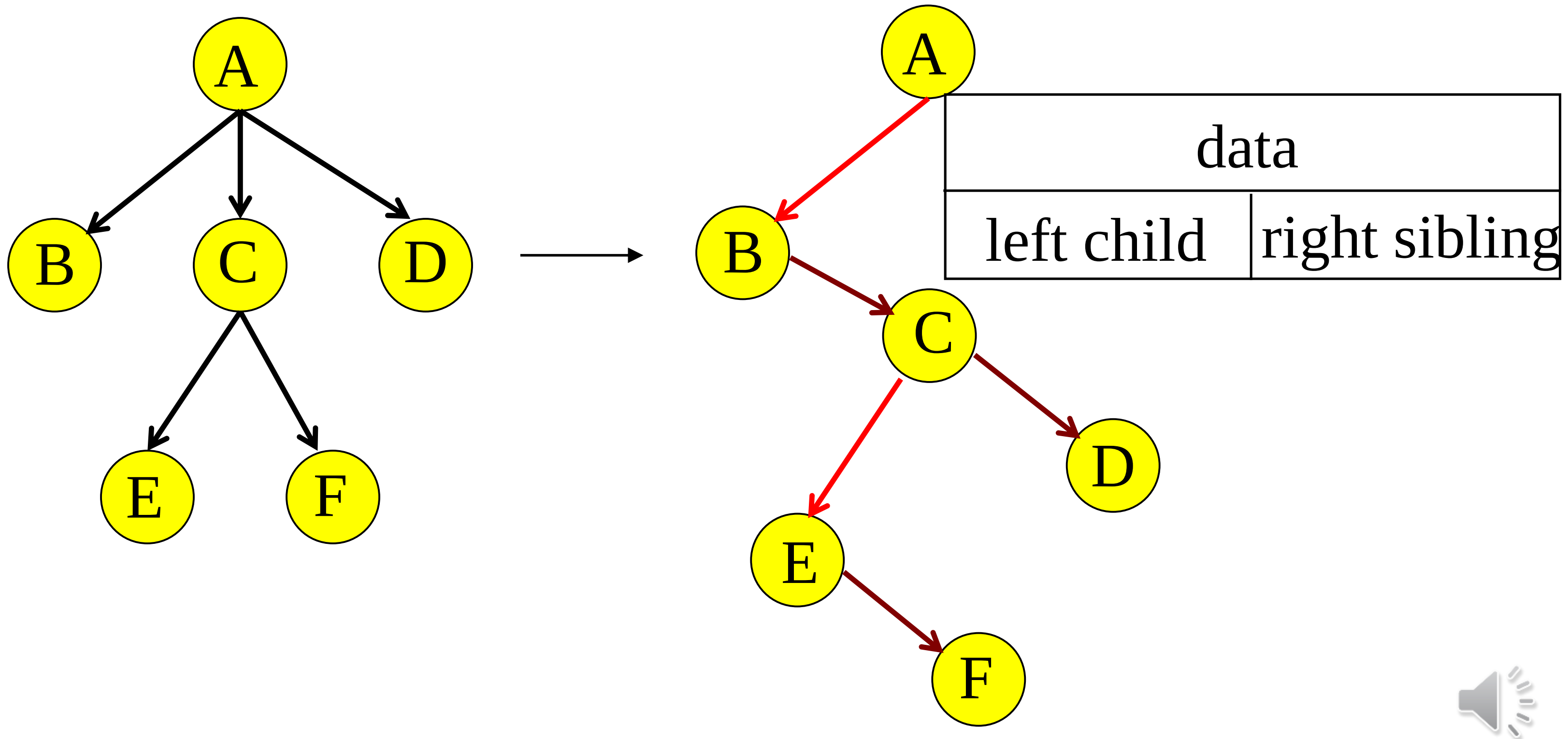
data	
left child	right sibling





Introduction(8/8)

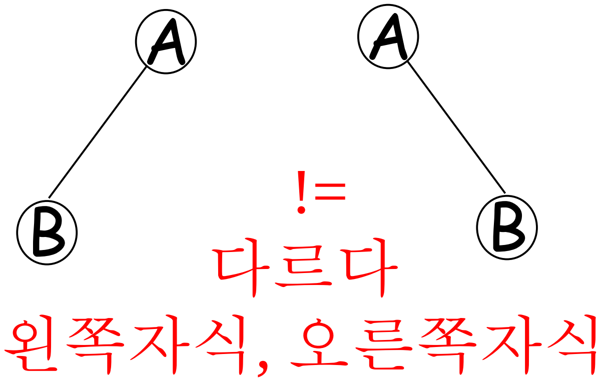
- Each node has 2 pointers: one to its first child and one to next sibling
- General tree일반트리-> binary tree이진트리



Binary Tree(이진 트리)

Binary Trees이진트리 (1/9)

- Binary trees are characterized by the fact that any node can have at most two branches이진트리는 최대 두 개의 가지를 가짐
- **Definition** (recursive):
  - A *binary tree* is a finite set of nodes that is either empty(공백) or consists of a root and two disjoint binary trees called the left subtree and the right subtree
  - ***Binary tree* = *root + left subtree+ right subtree***
    이진트리=루트+왼쪽서브트리+오른쪽서브트리
- Thus the left subtree and the right subtree are distinguished(구별된다)

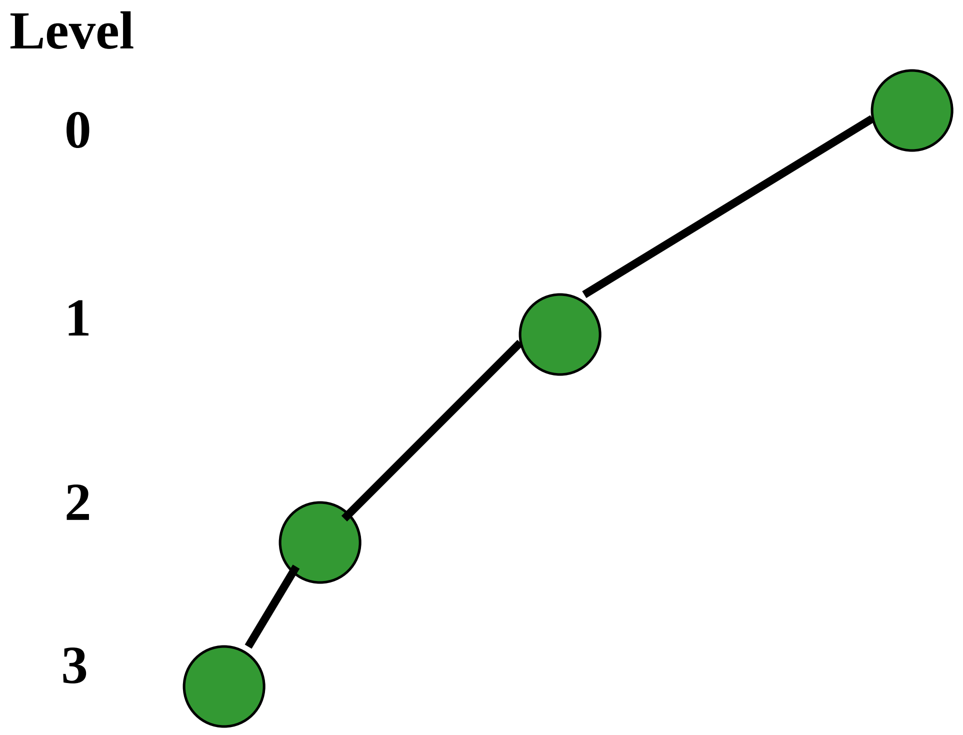


- Any tree can be transformed into binary tree
  - 어떤 일반트리도 이진트리로 변환가능하다.
  - By left child-right sibling representation (왼쪽큰아들, 오른쪽 남매)

Minimum Number Of Nodes

최소노드의 개수

- Minimum number of nodes in a binary tree whose height is h.(높이가 h인 트리)
- At least one node at each of first h levels.각 레벨에 하나씩만 노드가 있다면

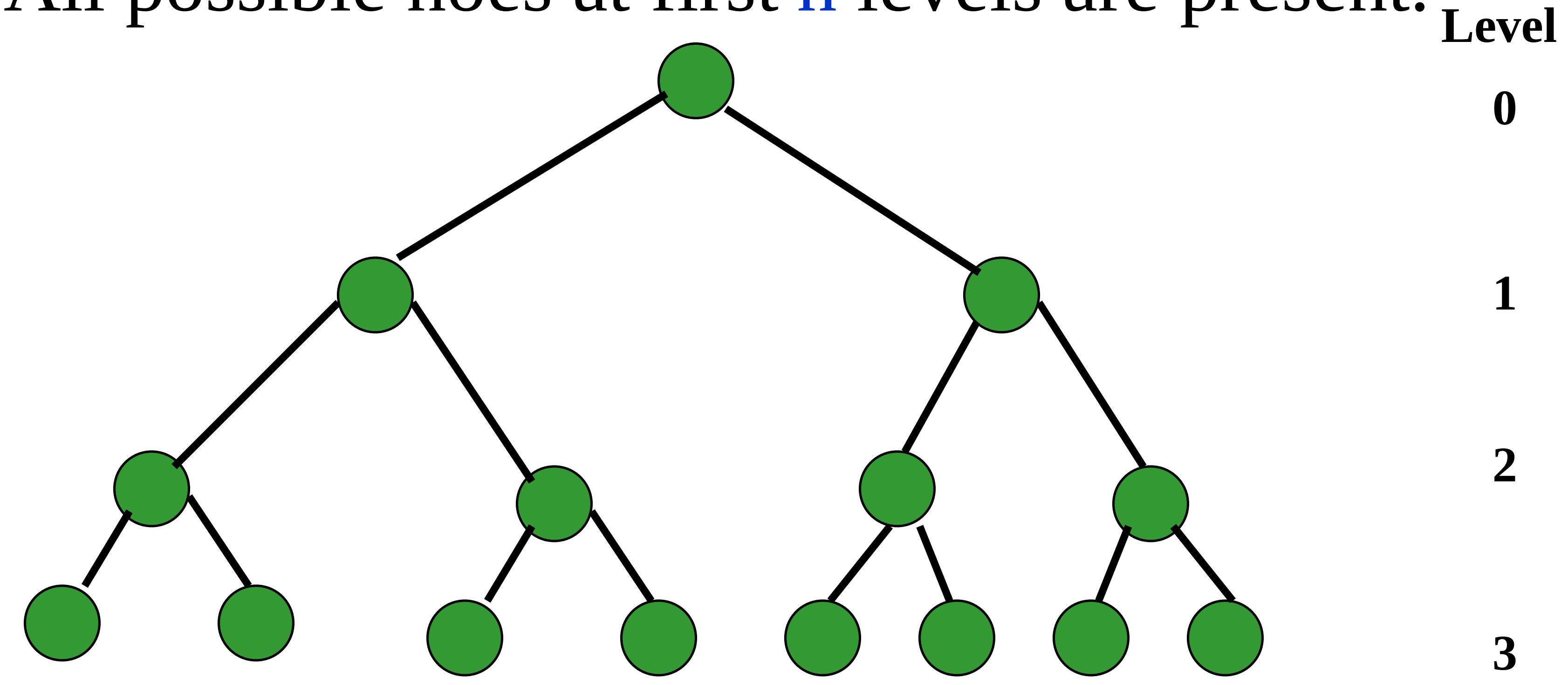


minimum number of nodes is h+1(최소노드의 수는 h+1)

Maximum Number Of Nodes

최대노드의 수

- All possible noes at first h levels are present.



Maximum number of nodes

$= 1 + 2 + 4 + 8 + \ldots + 2^h$

$= 2^{h+1} - 1$

Number Of Nodes & Height

- Let n be the number of nodes in a binary tree whose height is h. 높이가 h이고 노드의 개수가 n개 있다면
- $h + 1 <= n <= 2^{h+1} – 1$ (최소노드수 <= n <= 최대노드수)
- $\log_2(n+1) <= h+1 <= n$(위의 식을 변형했을경우)

Binary Trees (3/9)

- Three special kinds of binary trees:특별한 이진트리들
  (a) *skewed tree*, (경사이진트리,편향이진트리)
  (b) *full binary tree (포화 이진 트리)*
  (c) *complete binary tree (완전 이진 트리)*

Binary Trees (4/9)

- Definition:
  - A *full binary tree(포화이진트리)* of depth *k* is a binary tree of depth *k* having 2^k-1 nodes, k ≥ 0. (2^k-1개의 노드를 가진 깊이 k의 트리)
  - A binary tree with *n* nodes and depth *k* is complete(완전) *iff* its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth *k*.(완전이진트리정의) 포화이진트리에서 부여된 노드번호들과 완전히 일치하는 이진트리(다음페이지에서 번호부여 나옴)
  - The height of a complete binary tree (n개의 노드를 가진완전이진트리의 높이)

with *n* nodes is $\lceil \log_2(n+1) \rceil$

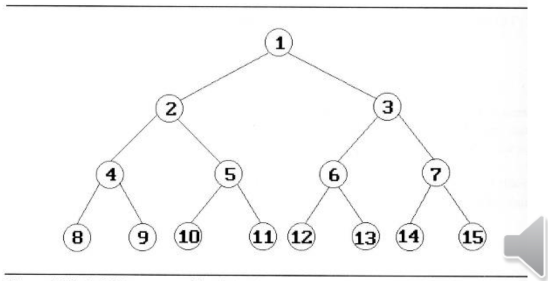
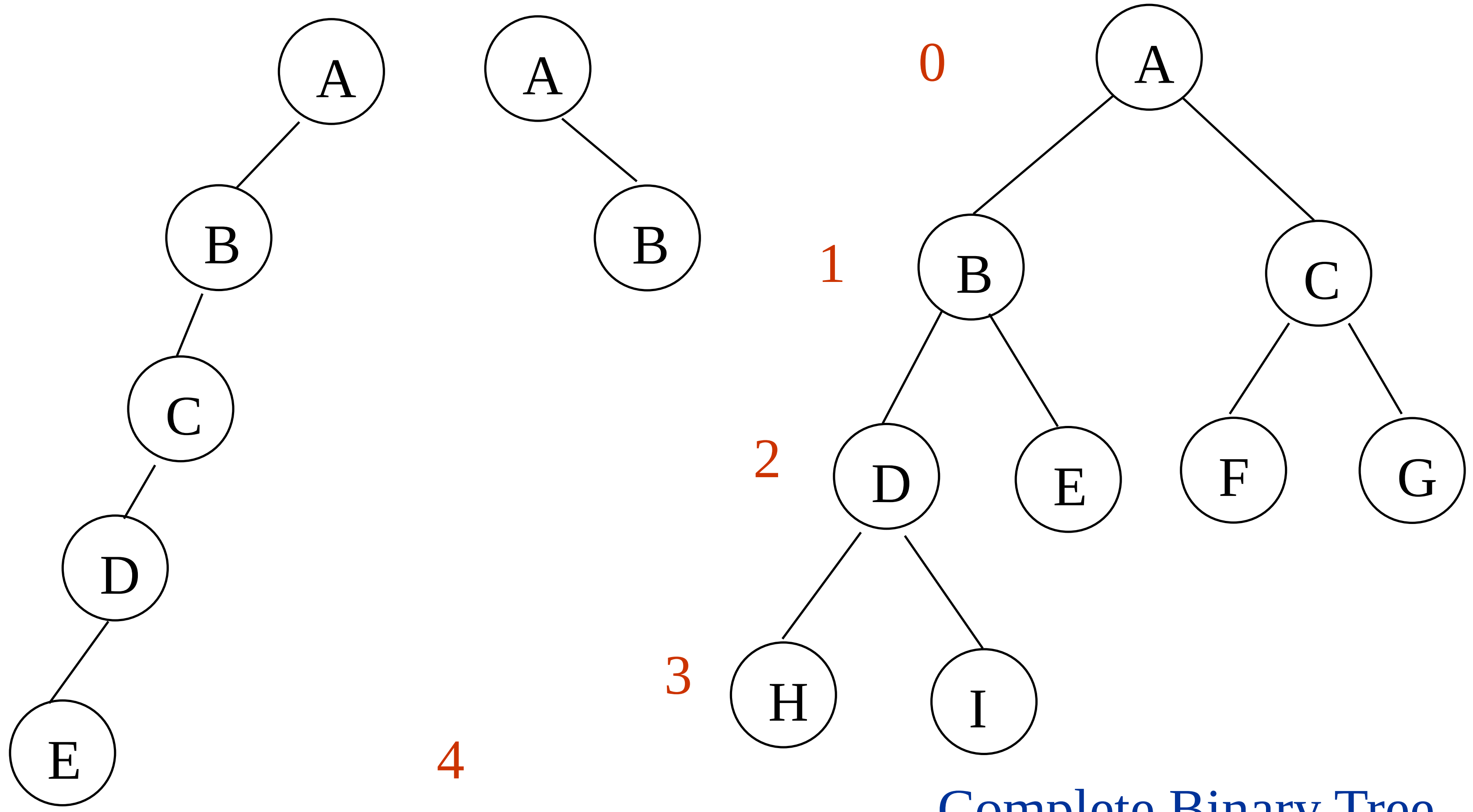


Figure 5.10: Full binary tree of depth 4 with sequential node numbers

경사이진트리와 완전이진트리비교

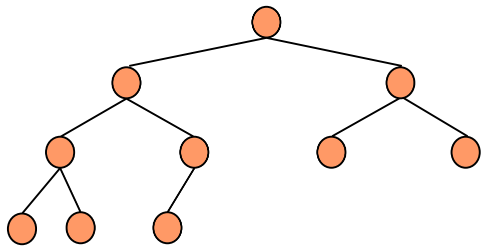


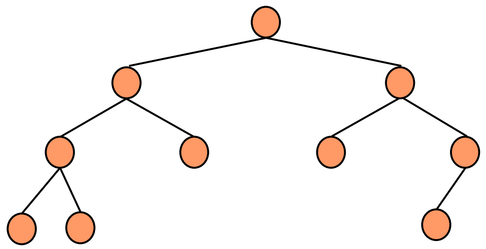
Skewed Binary Tree
경사이진트리

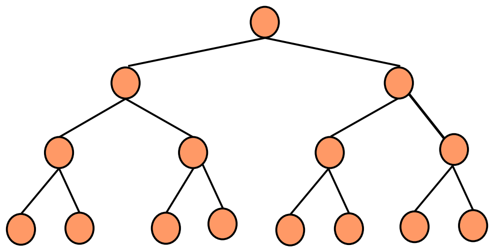
Complete Binary Tree
완전이진트리

Full & Complete Trees

- Before we look at how we represent trees let's just look and one more definition
- A full binary tree(포화이진트리) is a tree in which all the nodes except the leaves have two children(포화이진트리는 나뭇잎노드를 제외한 모든 노드들이 2개의 자식노드를 가짐
- A complete binary tree(완전이진트리) is a tree that is either full or full up to the last but one level, and have all the nodes in the bottommost level shifted to the far left.

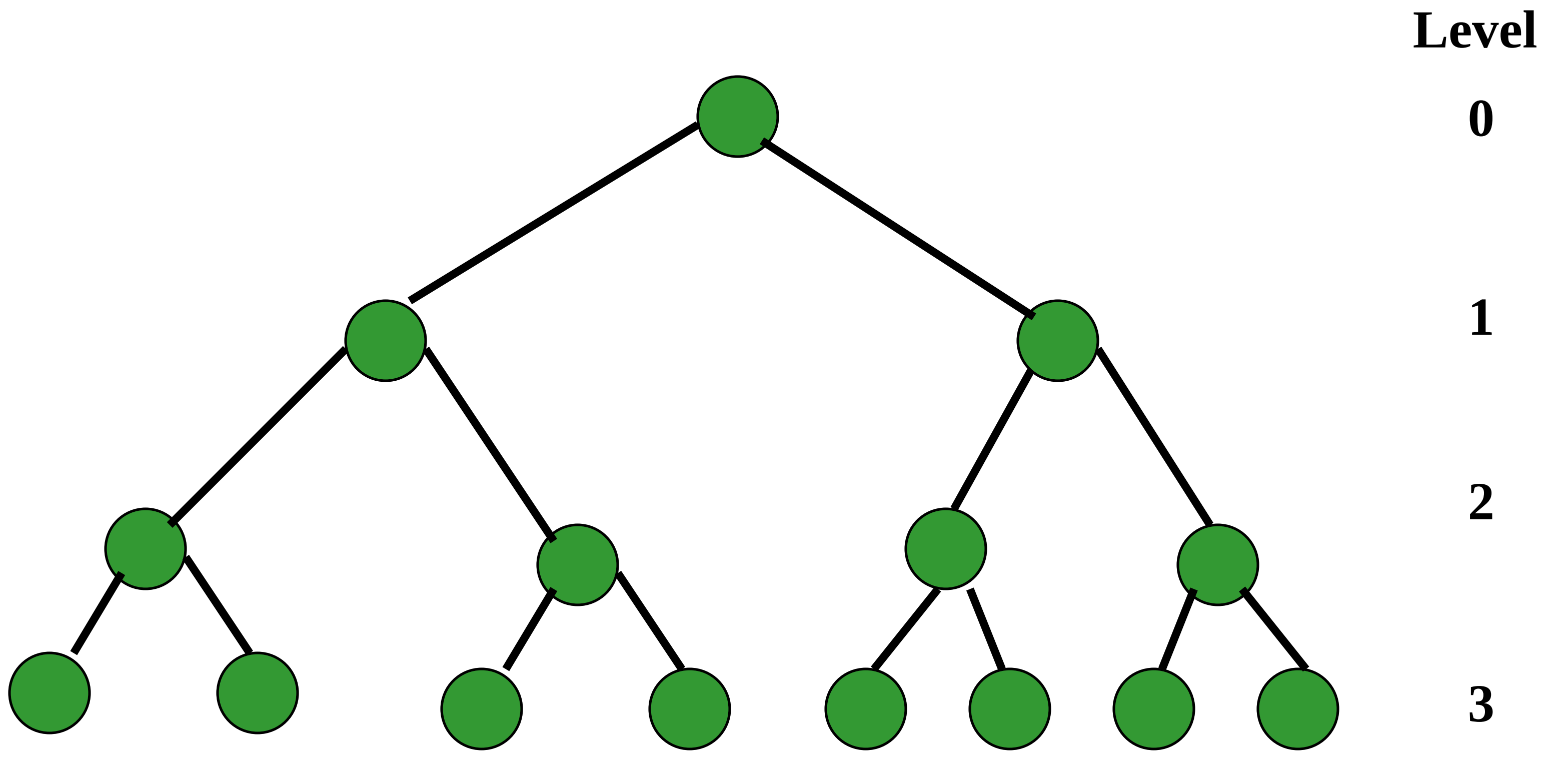

A complete binary tree
완전이진트리


An incomplete binary tree
완전이진트리가 아님


A full binary tree
포화이진트리이자
완전이진트리

Full Binary Tree
포화이진트리

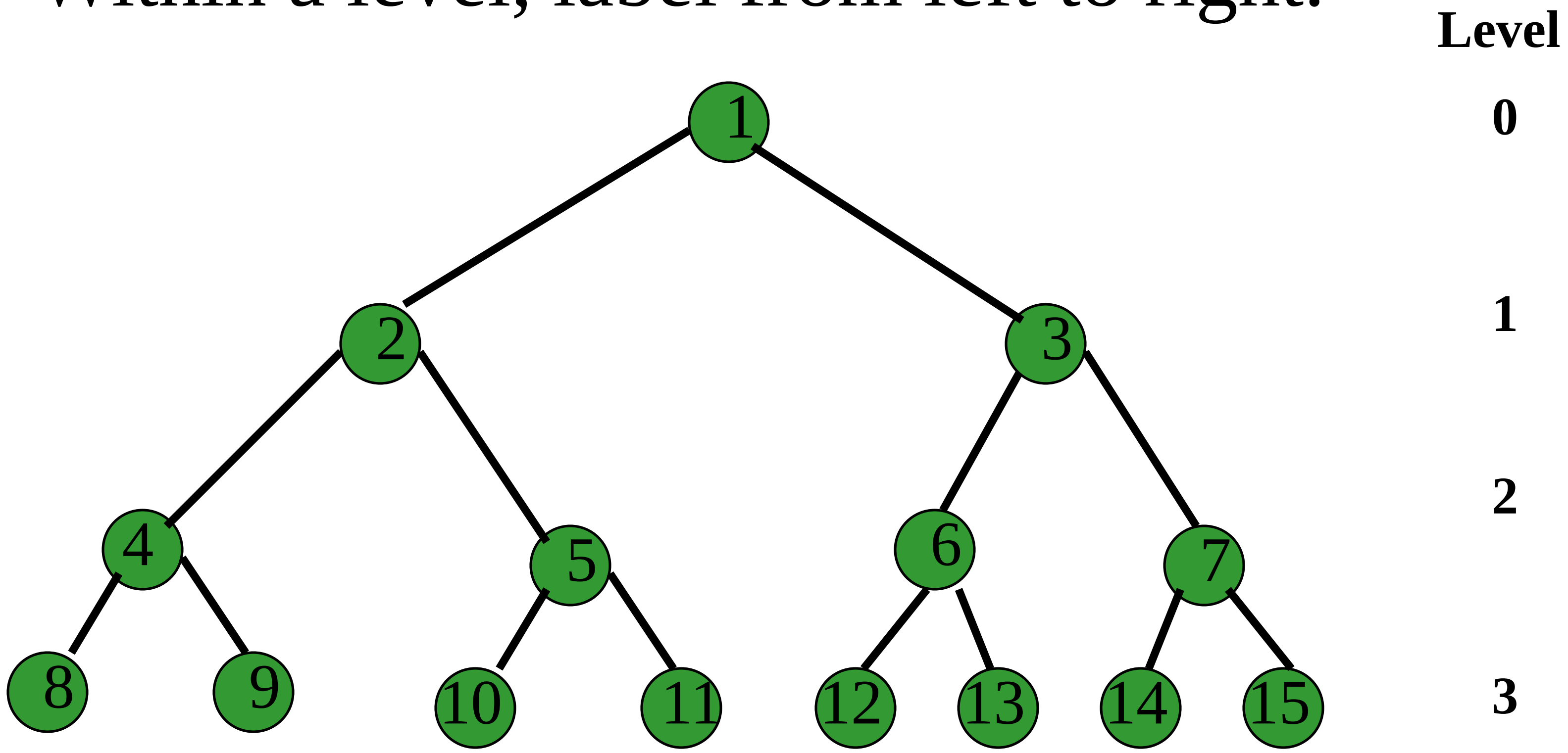
- A full binary tree of a given height k has $2^{k+1}-1$ nodes.



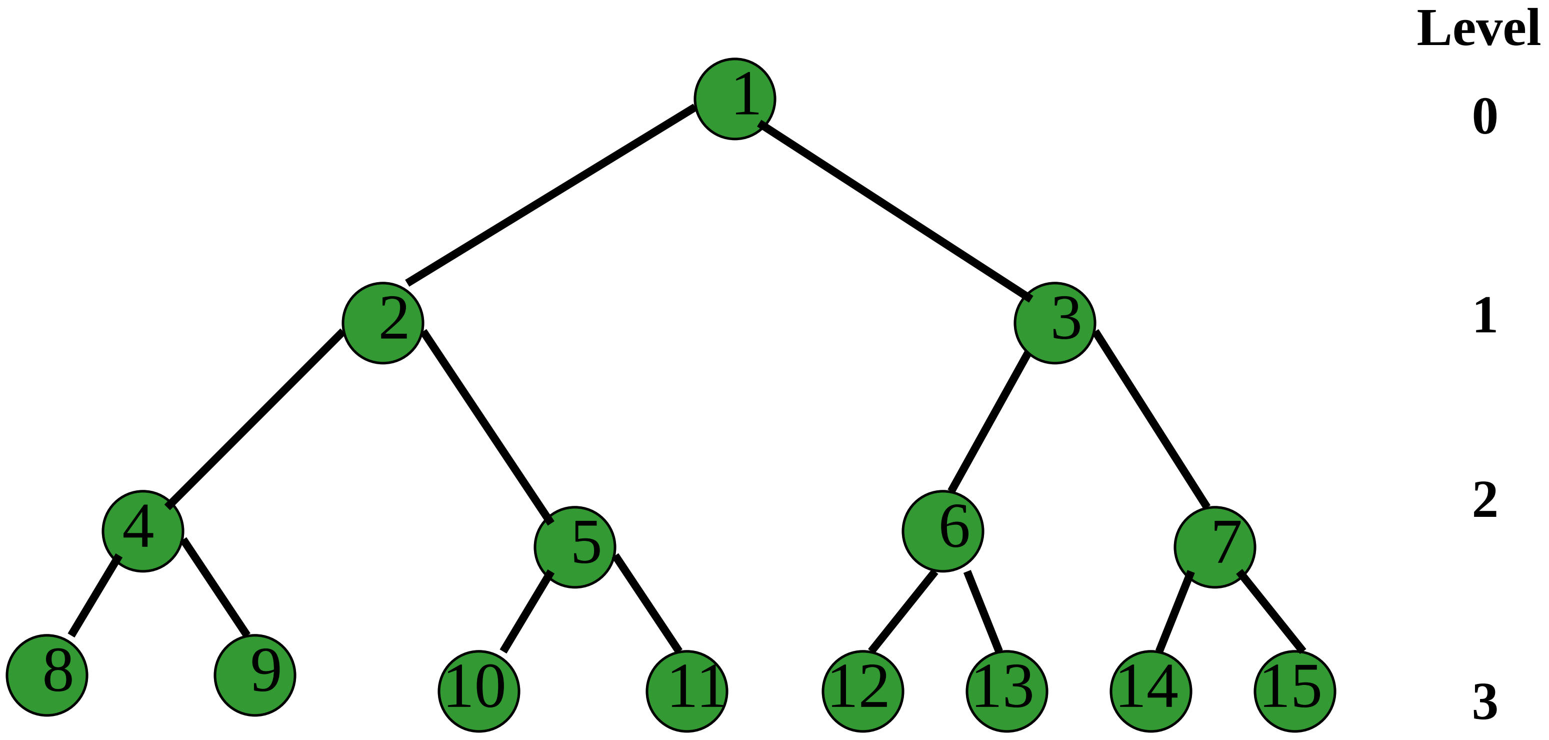
Height 3 full binary tree.

Labeling Nodes In A Full Binary Tree
포화이진트리의 번호부여

- Label the nodes 1 through $2^{k+1} – 1$.
- Label by levels from top to bottom.
- Within a level, label from left to right.

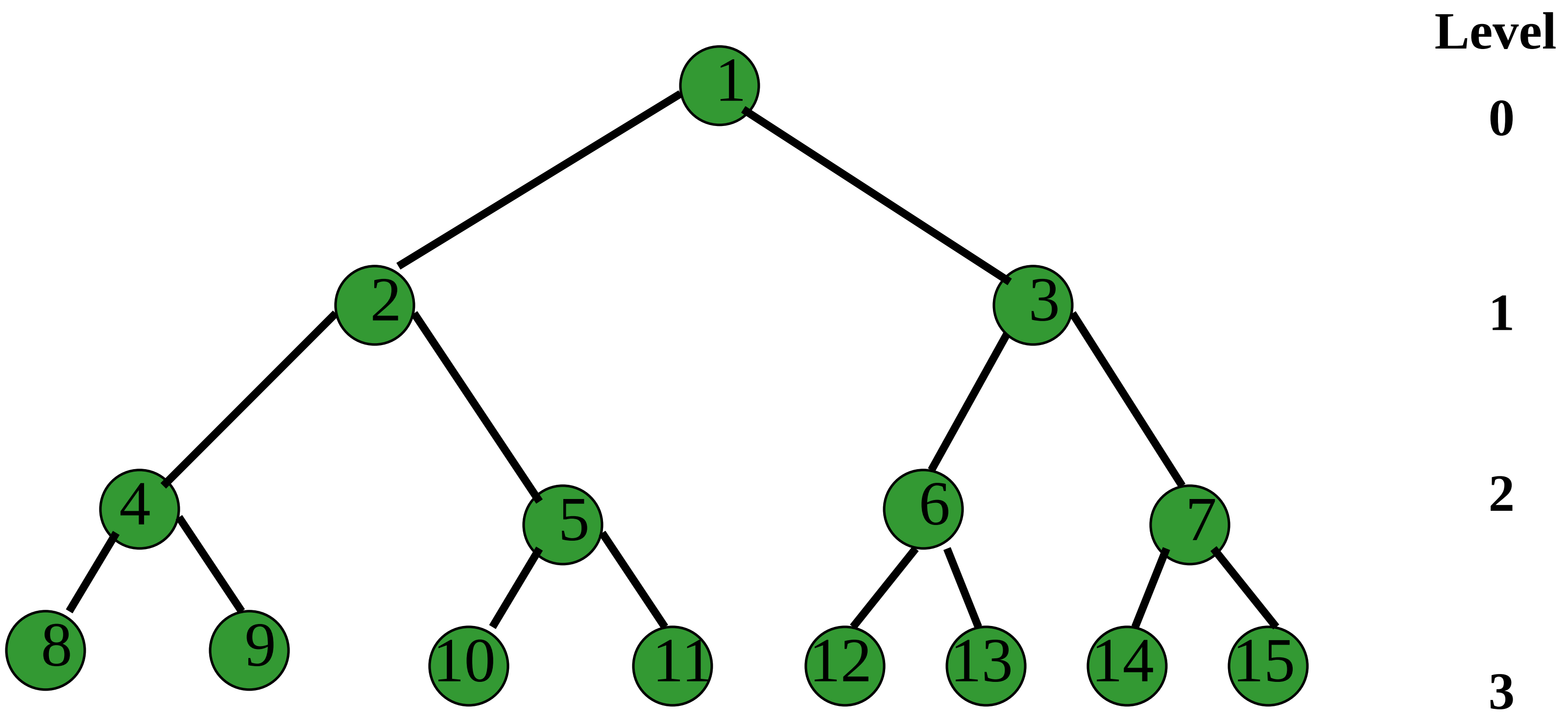


Node Number Properties
노드번호와 관련된 특징



- Parent of node i is node i / 2, unless i = 1. 특정노드i의 아버지는 i / 2
- Node 1 is the root and has no parent. 1번은 루트

Node Number Properties
노드번호와 관련된 특징



- Left child of node i is node 2i, unless 2i > n, where n is the number of nodes. 노드 i의 왼쪽자식은 2i, 오른쪽자식은 2i+1,
- If 2i > n, node i has no left child. (만일 2i > n, 노드 i는 자식없음)

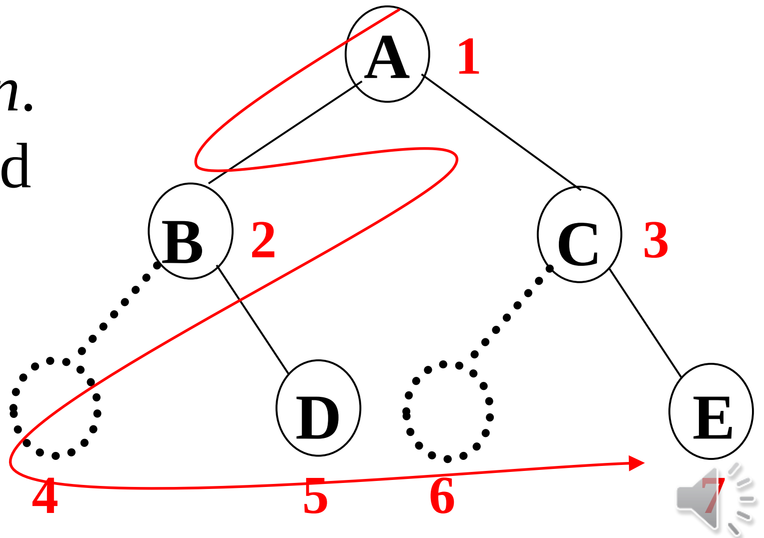
Binary Trees-array (6/9)(배열표현)

- Binary tree representations (using array)
  - If a complete binary tree with n nodes is represented sequentially, then for any node with index i, $1 \le i \le n$, we have

1. *parent(i)* is at $\lfloor i/2 \rfloor$ if $i \ne 1$.
   If $i = 1$, i is at the root and has no parent.
2. *LeftChild(i)* is at $2i$ if $2i \le n$.
   If $2i > n$, then i has no left child.
3. *RightChild(i)* is at $2i+1$ if $2i+1 \le n$.
   If $2i+1 > n$, then i has no left child

[1]	[2]	[3]	[4]	[5]	[6]	[7]
A	**B**	**C**	**—**	**D**	**—**	**E**

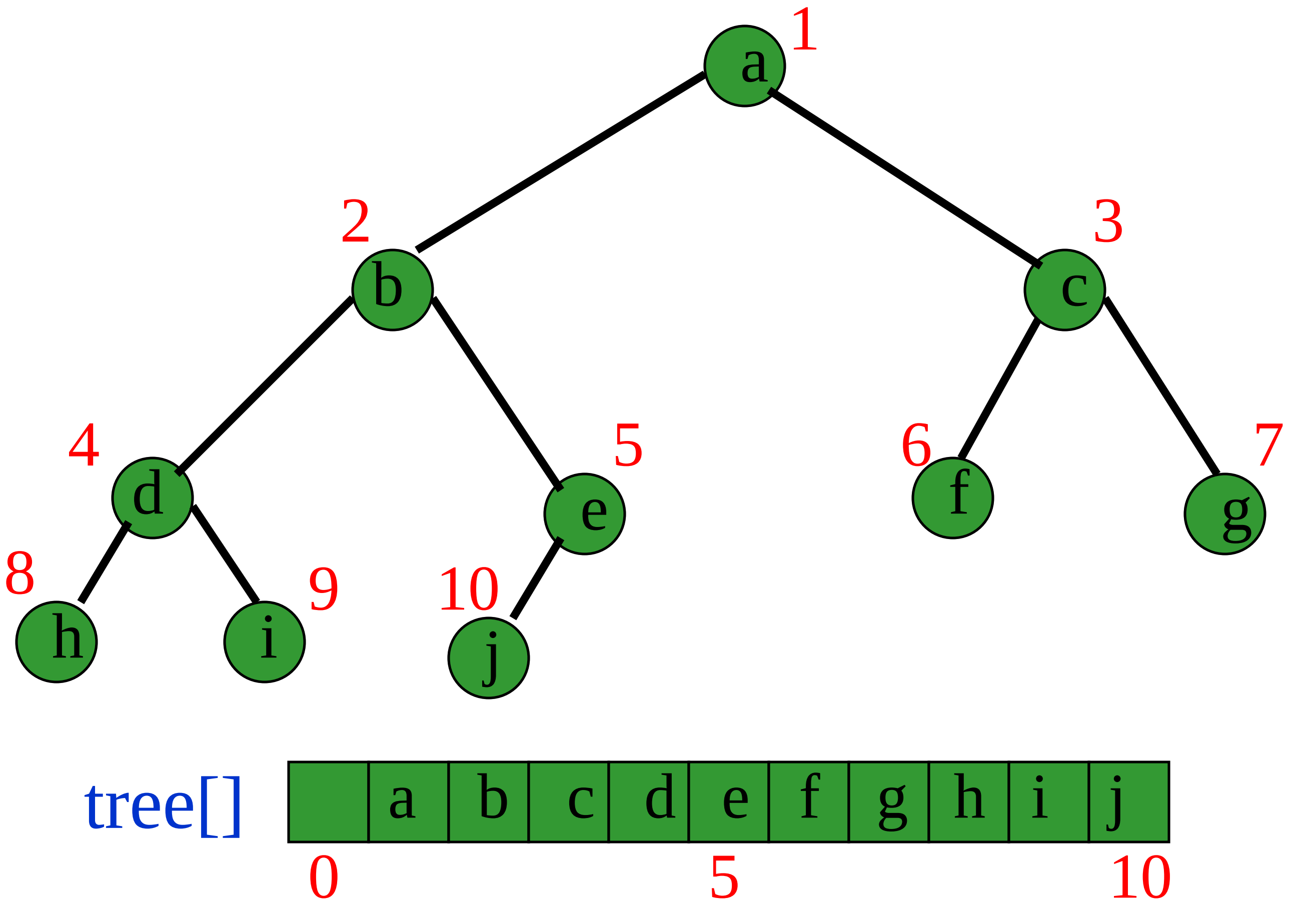
Level 0 ([1]) **Level 1** ([2]–[3]) **Level 2** ([4]–[7])



Array Representation

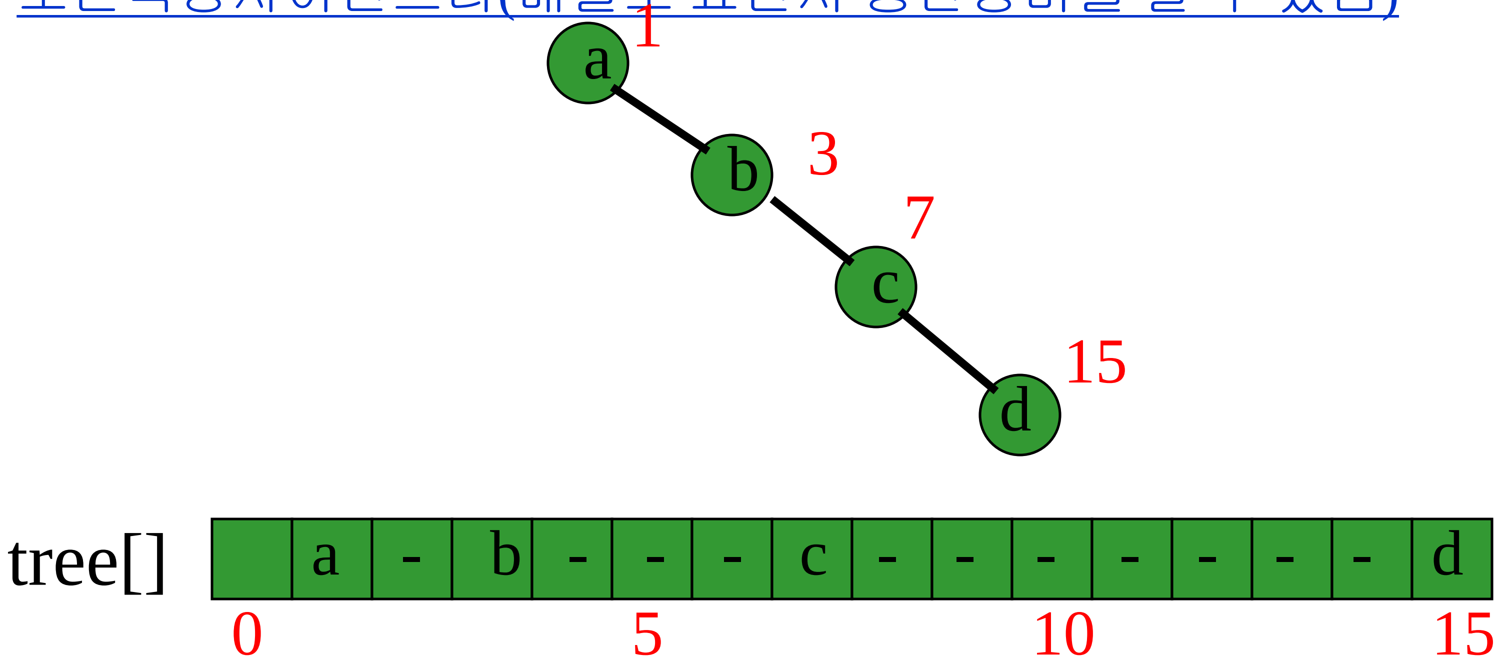
배열로 이진트리표현

- Number the nodes using the numbering scheme for a full binary tree. The node that is numbered i is stored in tree[i].



Right-Skewed Binary Tree

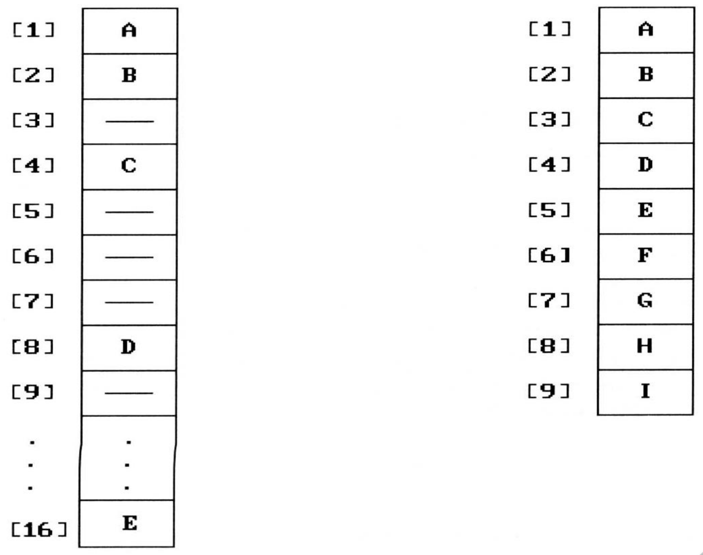
오른쪽경사이진트리(배열로 표현시 공간낭비를 알 수 있음)



- An n node binary tree needs an array whose length is between n+1 and 2^n. 이진트리를 배열로 표현하려면 n+1과2^n 사이의 방이필요

Binary Trees (8/9)-array-Adv장점

- Binary tree representations (using array배열사용)
  - Simplicity(단순함)
  - Can be applied to any tree(어떤트리에도 적용)
  - Best for complete binary tree (완전이진트리에 최적)
  - no need to store left and right pointers in the nodes ➔ save memory(왼쪽 오른쪽 끈이 필요없다→메모리절약)
  - Direct access to nodes(노드k에 직접접근가능): to get to node k, access A[k]



[1]	A
[2]	B
[3]	—
[4]	C
[5]	—
[6]	—
[7]	—
[8]	D
[9]	—
. . .	. . .
[16]	E

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

Figure 5.11: Array representation of binary trees of Figure 5.9

Binary Trees -array(7/9)-disadv.단점

- Binary tree representations (using array배열사용)
  - Waste spaces(공간낭비): in the worst case, a skewed tree of depth *k* requires 2^k-1 spaces. Of these, only *k* spaces will be occupied (특별히 경사이진트리)
  - Insertion or deletion of nodes from the middle of a tree requires the movement of potentially many nodes to reflect the change in the level of these nodes

삽입삭제시 많은원소 이동가능성

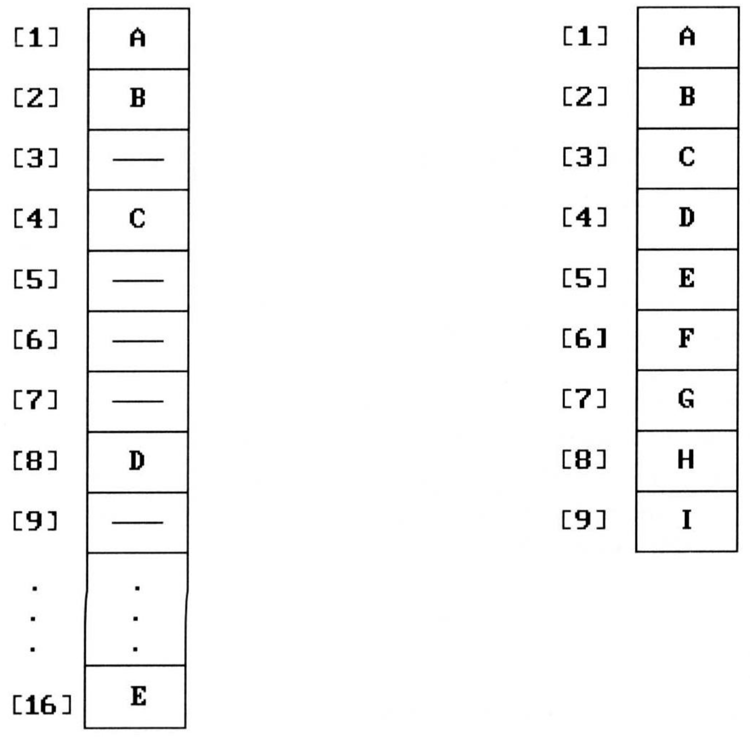


Figure 5.11: Array representation of binary trees of Figure 5.9

Binary Trees -link(8/9)

이진트리를 연결리스트로 표현

- Binary tree representations (using link 링크, 포인터)

```
class TreeNode{
        Object data;
        TreeNode left;   link, 포인터, 끈
        TreeNode right; link, 포인터, 끈
```

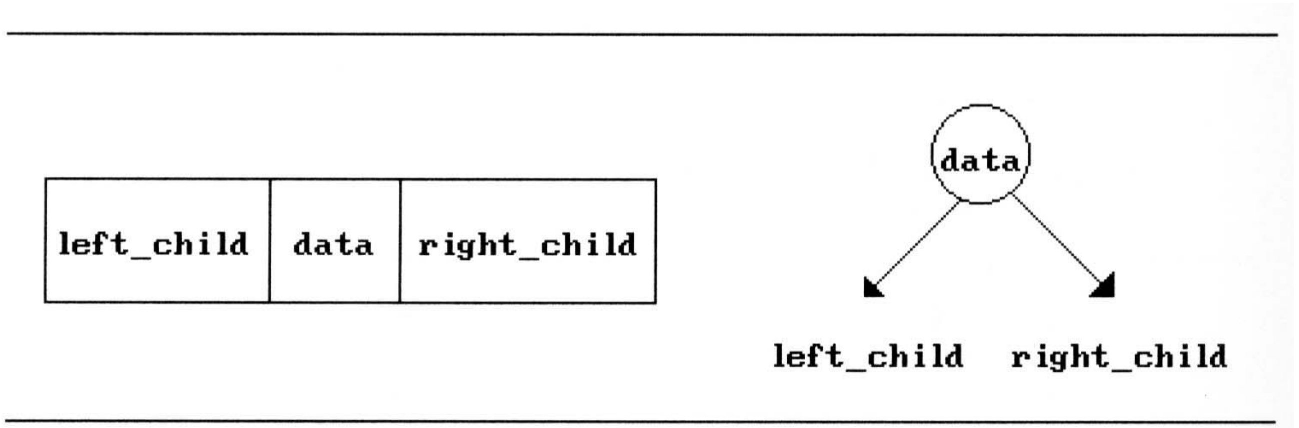
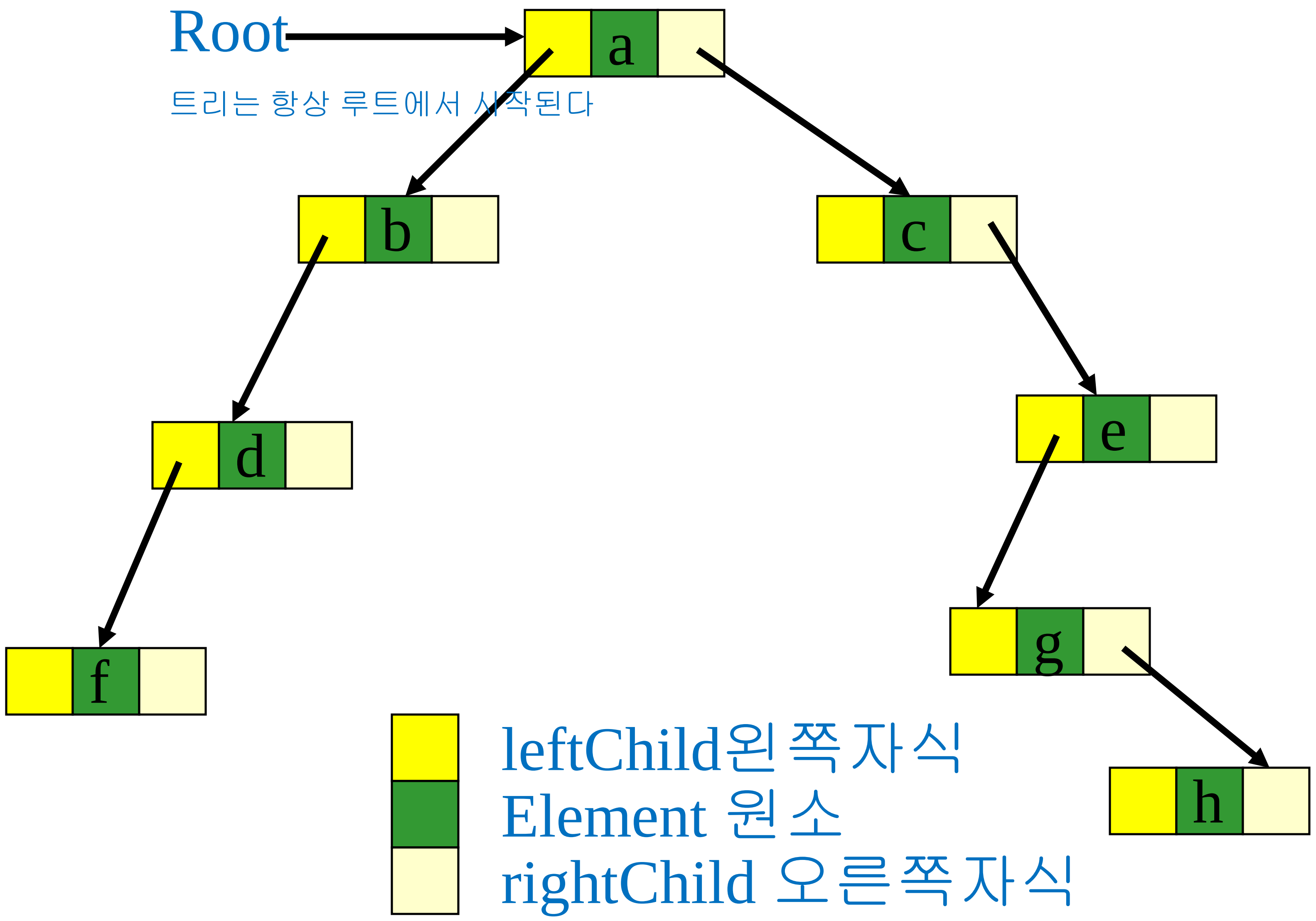


Figure 5.12: Node representation for binary trees

Binary tree-link-adv/disadv?(장점?단점?)

- 연결리스트의 장점
  - Dynamic size 크기가 다양하다.
  - Good for skewed binary tree 경사이진트리표현에 최적
  - Easy to insertion/deletion 삽입과 삭제가 쉽다.
- 연결리스트의 단점
  - No direct access to node. 원소에 직접접근이 허용되지 않는다.
  - Additional storage required. 왼쪽 오른쪽 포인터를 위한 추가적인 공간필요

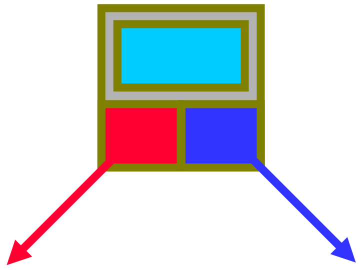
Linked Representation Example



Traversing a Binary Tree (이진트리순회방문) 모든 노드를 한번씩 방문

The Scenario시나리오

- **Imagine we have a binary tree**
- **We want to traverse the tree**
  - **It's not linear**
  - **We need a way to visit all nodes**



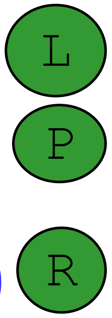
- **Three things must happen:**
  - **Deal with the entire left sub-tree(왼쪽서브트리)** L
  - **Deal with the current node 방문한노드에서작업** P
  - **Deal with the entire right sub-tree(오른쪽서브트리)** R

Summary 요약

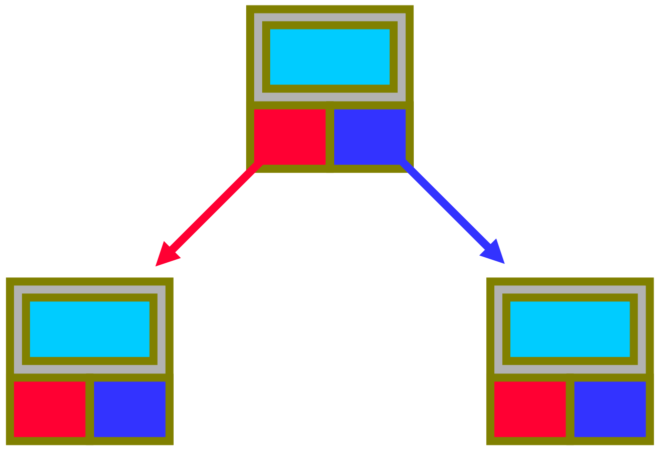
- **An In-Order(중위순회) traversal visits every node**
  - **Recurse left first(왼쪽자식으로 이동)**
  - **Do something with current node(특정작업)**
  - **Recurse right last(오른쪽자식으로 이동)**
- **The "left, current, right" logic is repeated recursively at every node.** 이 작업을 재귀적으로 수행

Outline of In-Order Traversal 중위순회

- **Three principle steps:**3원칙
  - **Traverse Left(**왼쪽으로간다**)**
  - **Do work (Current)-Data(**인쇄**)**
  - **Traverse Right(**오른쪽으로간다**)**



- **Work can be anything(**어떤 작업도 가능**)**



- Traverse the tree “In order”:
  - Visit the tree’s left sub-tree L
  - Visit the current and do work P
  - Visit right sub-tree R

Inorder traversal(중위순회)

```
inorder(T)
    if (T ≠ null) then {
            inorder(T.left);        L
            visit T.data;           P
            inorder(T.right);       R
    }
end inorder()
```

Preorder traversal(전위순회)

- In preorder(전위순회), the root is visited *first*
- Here's a preorder traversal to print out all the elements in the binary tree:

```
preorder(T)
    if (T ≠ null) then {
            visit T.data;          P
          preorder(T.left);        L
          preorder(T.right);       R
    }
end preorder()
```

Postorder traversal(후위순회)

- In postorder(후위순회), the root is visited *last*
- Here’s a postorder traversal to print out all the elements in the binary tree:

```
postorder(T)
    if (T $\neq$ null) then {
        postorder(T.left);      L
        postorder(T.right);     R
        visit T.data;           P
    }
```

한국학생들에게

- 최대한 이해가 쉽게 그림 등을 많이 넣었습니다
- 영어가 이해가 안되면 다른 강좌를 수강하기를 권합니다.
- 강의가 이해가 안되면 언제든지 연락바랍니다.