CHAPTER 5

TREES

5.1 Introduction

5.1.1 Terminology

- A **tree** is a finite set of one or more nodes such that
  1) There is a specially designated node called **root**
  2) The remaining nodes are partitioned into n (≥ 0) disjoint set $T_1, \ldots, T_n$, where T_i: the **subtrees** of the root

This is a recursive definition; Every item (node) in a tree is the root of some subtree

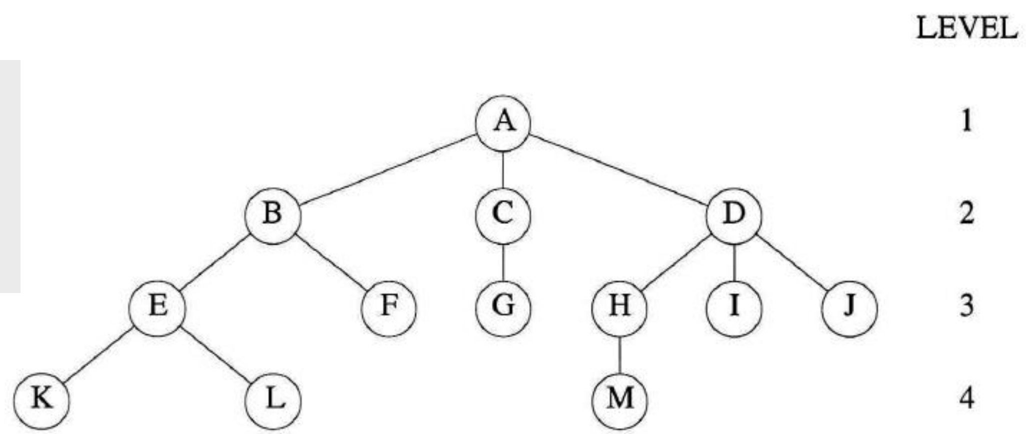


Figure 5.2: A sample tree

- Some Terminology
  - node: …
  - degree of a node: ….
  - terminal (or leaf) nodes :
  - nonterminal nodes:
  - children, parent, siblings :
  - degree of a tree
    - max{degree of the nodes}
  - ancestors : …
  - level of a node : …
  - height (depth) of a tree
    - max{level of the nodes}

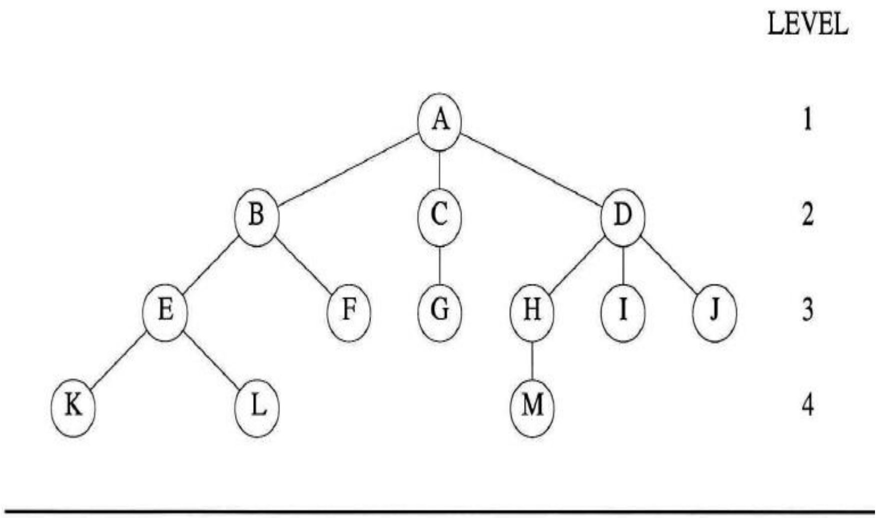
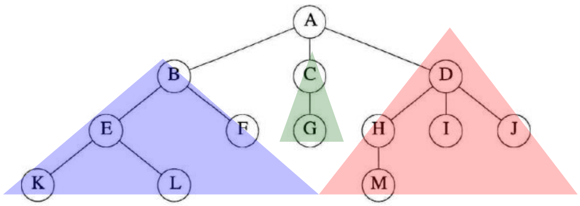


Figure 5.2: A sample tree

5.1.2 Representation of Trees



1) List Representation

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

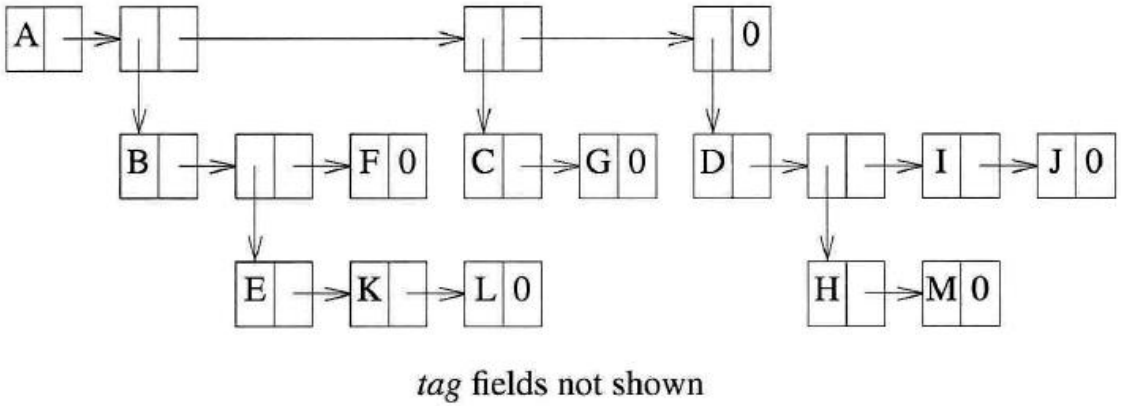


Figure 5.3: List representation of the tree of Figure 5.2

- Node representation

DATA	CHILD 1	CHILD 2	···	CHILD *k*

Figure 5.4: Possible node structure for a tree of degree *k*

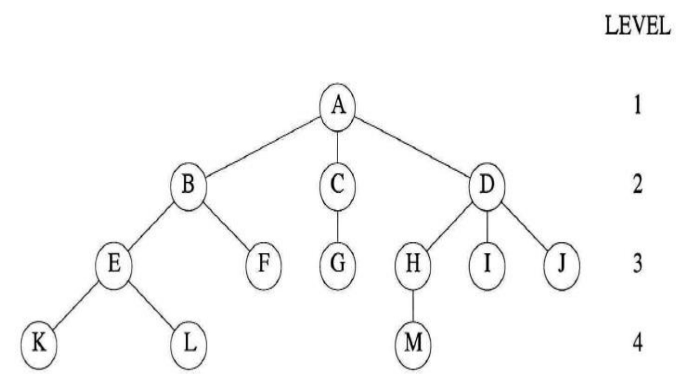


Figure 5.2: A sample tree

  - # of zero fields: $nk-(n-1)$
  - $n(k-1)+1$ of the nk child fields are 0
  - Wasteful of space

이만큼이 0이다 → 메모리 낭비

2) Left Child-Right Sibling(LCRS) Representation

이진 트리 트리

data	
left child	right sibling

Figure 5.5: Left child-right sibling node structure

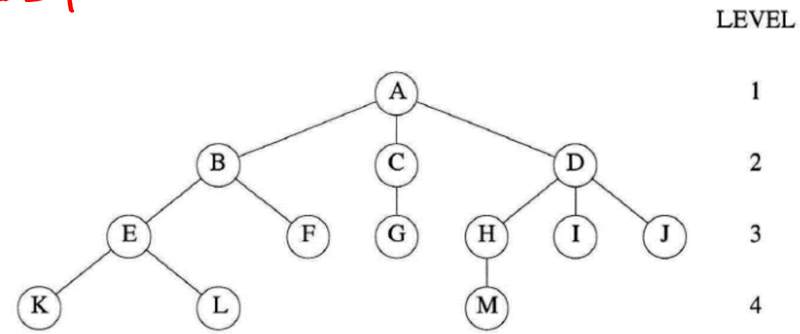


Figure 5.2: A sample tree

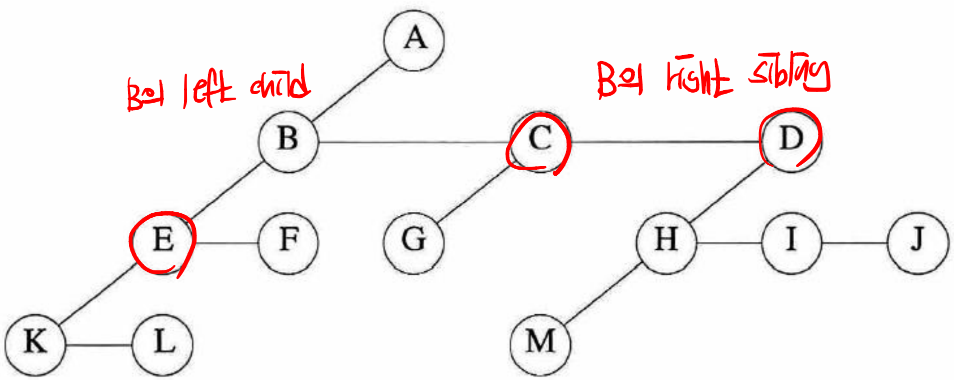


Figure 5.6: Left child-right sibling representation of tree of Figure 5.2

3) Representation as a Degree Two Trees

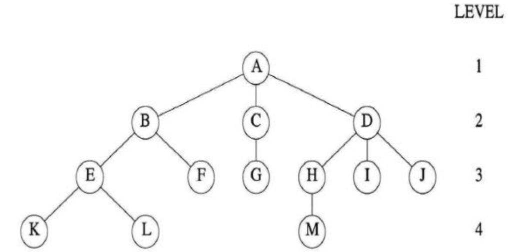
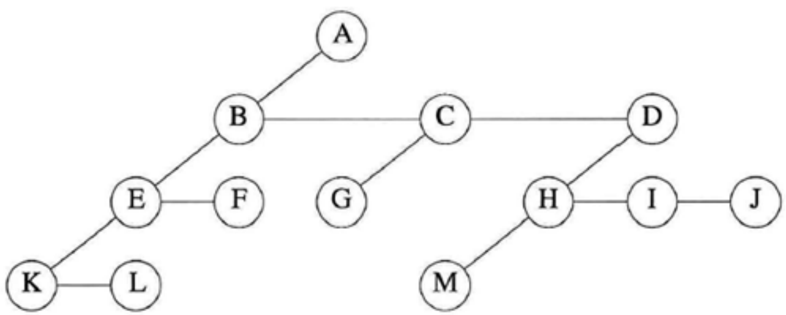
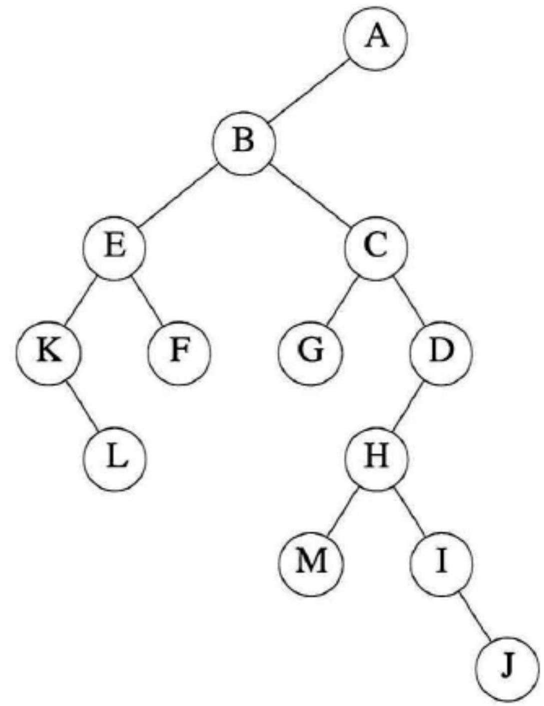


Figure 5.2: A sample tree

Data	
Left Child	Right Child





re 5.7: Left child-right child tree representation of tree of Figure 5.2

Rotate the right-sibling pointers in a LCRS tree clockwise by 45 degrees

Every tree can be transformed into a binary tree

항상 이진 트리로 바꿀 수 있음

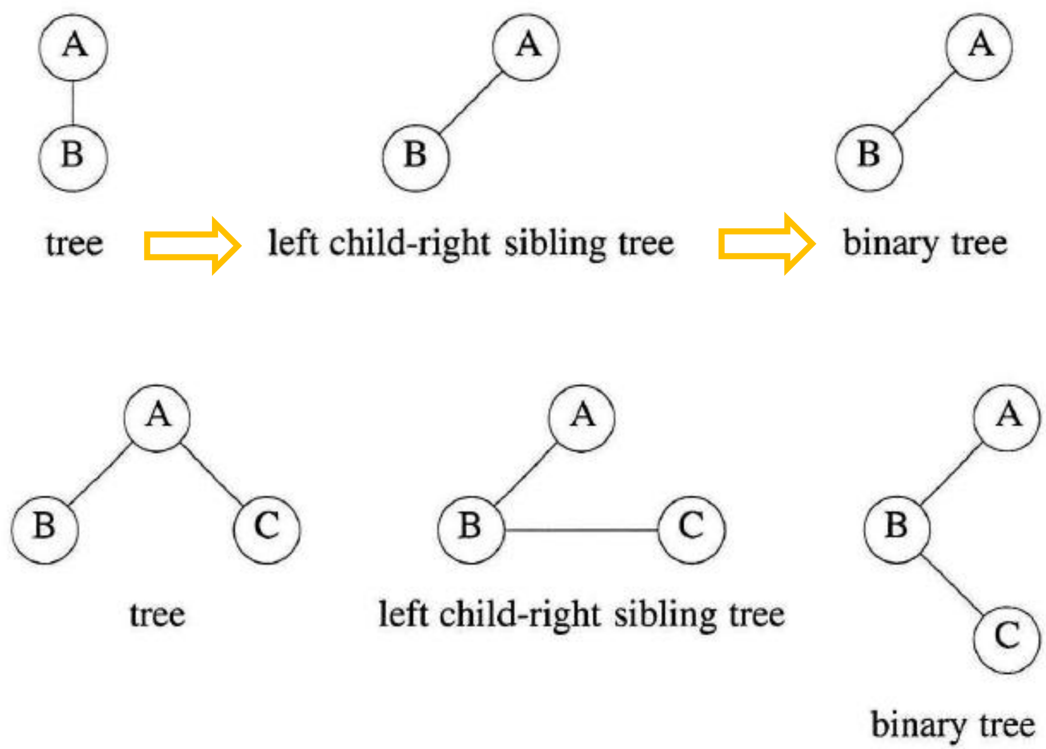
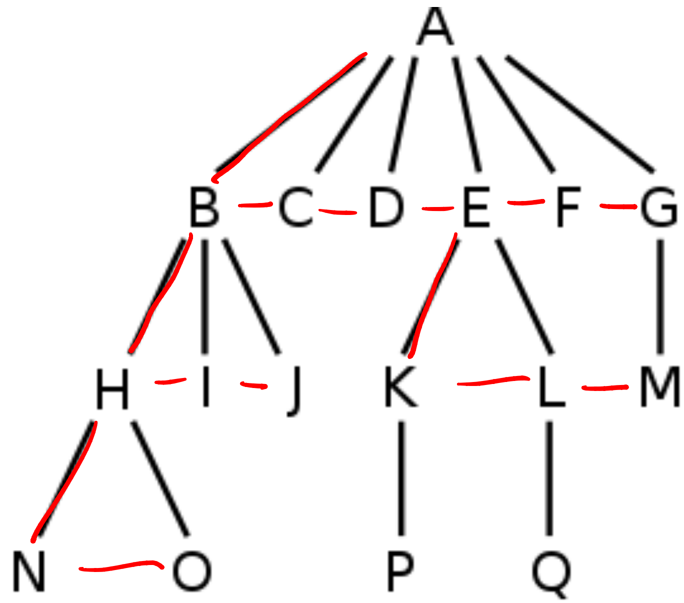
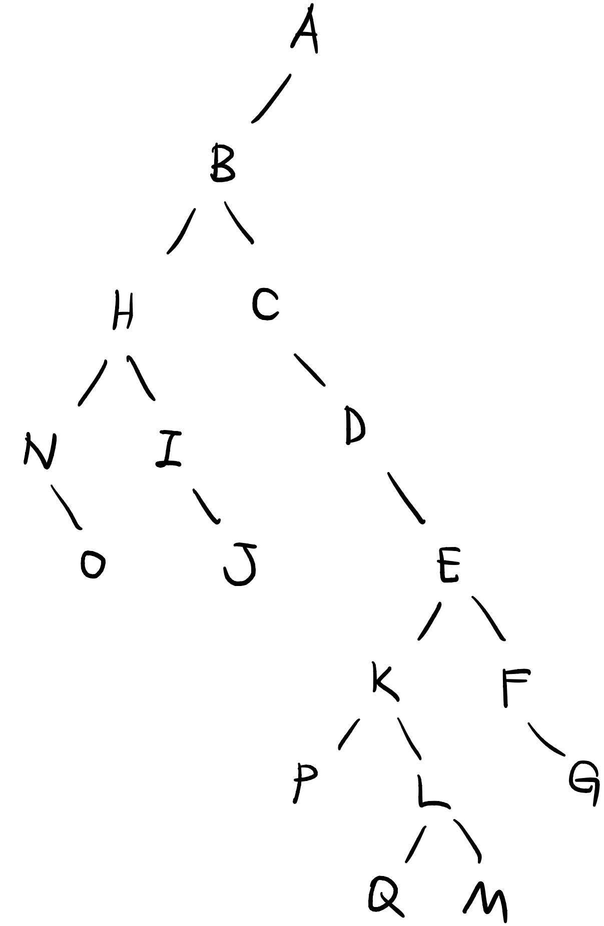


Figure 5.8: Tree representations


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q


A
B
H
C
N
I
D
O
J
E
K
F
P
L
G
Q
M

5.1 Introduction

5.2 Binary Trees

5.3 Binary Trees Traversals

5.4 Additional Binary Tree Operations

5.5 Threaded Binary Trees

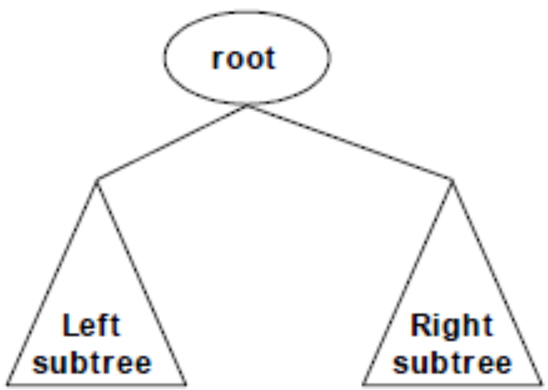
5.6 Heaps

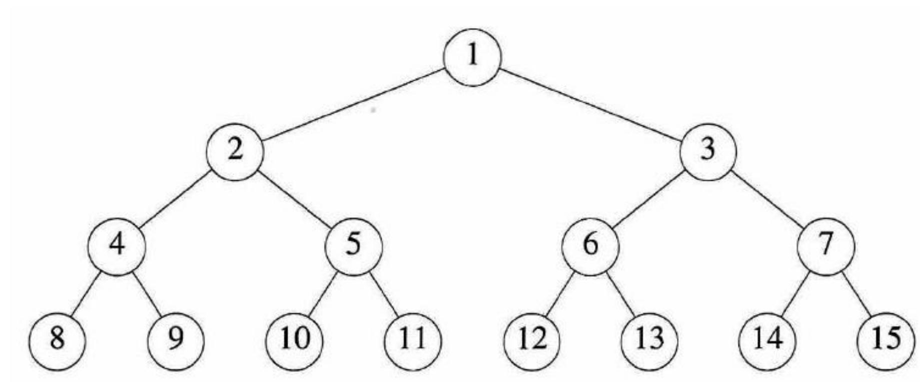
5.7 Binary Search Trees

5.2.1 The Abstract Data Type

→ 얘도 binary tree

- A **binary tree** is a finite set of nodes that is either **empty** **or** consists of a **root** and **two disjoint binary trees** (called the left subtree and the right subtree)





The degree of any given node must not exceed two

모든 node의 최대 degree가 2를 넘지않는 것

ADT *Binary_Tree* (abbreviated *BinTree*) is

objects: a finite set of nodes either empty or consisting of a root node, left *Binary_Tree*, and right *Binary_Tree*.

functions:

for all $bt, bt1, bt2 \in BinTree$, $item \in element$

BinTree Create()	::=	creates an empty binary tree
Boolean IsEmpty(*bt*)	::=	**if** (*bt* == empty binary tree) **return** *TRUE* **else return** *FALSE*
BinTree MakeBT(*bt*1, *item*, *bt*2)	::=	**return** a binary tree whose left subtree is *bt*1, whose right subtree is *bt*2, and whose root node contains the data *item*.
BinTree Lchild(*bt*)	::=	**if** (IsEmpty(*bt*)) **return** error **else return** the left subtree of *bt*.
element Data(*bt*)	::=	**if** (IsEmpty(*bt*)) **return** error **else return** the data in the root node of *bt*.
BinTree Rchild(*bt*)	::=	**if** (IsEmpty(*bt*)) **return** error **else return** the right subtree of *bt*.

ADT 5.1: Abstract data type *Binary_Tree*

- Differences between a **Tree** & a **Binary Tree**
  - There is no tree having zero nodes, but there is an empty binary tree
  - In a binary tree, we distinguish between the order of the children while in a tree we do not
  - e.g)

binary tree는 left child와 right child의 order를 구분한다

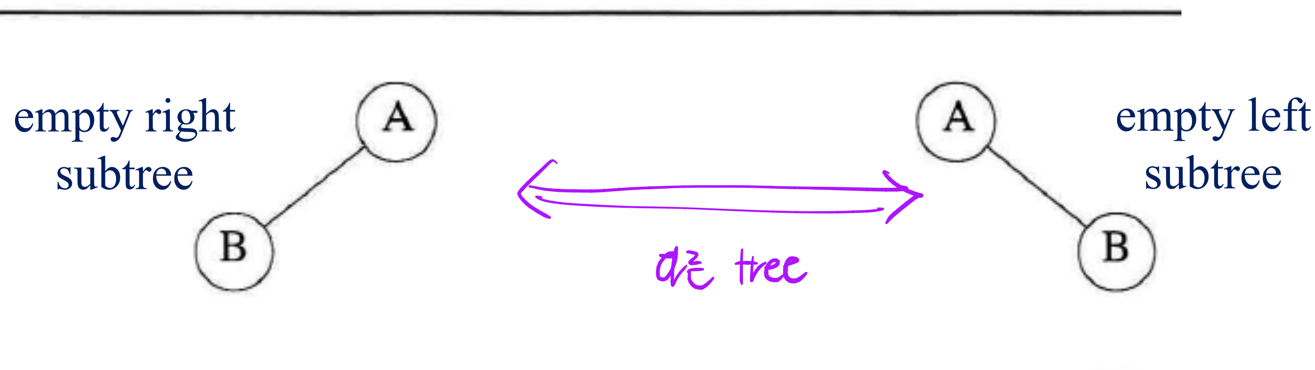


Figure 5.9: Two different binary trees

- Two special kinds of binary trees

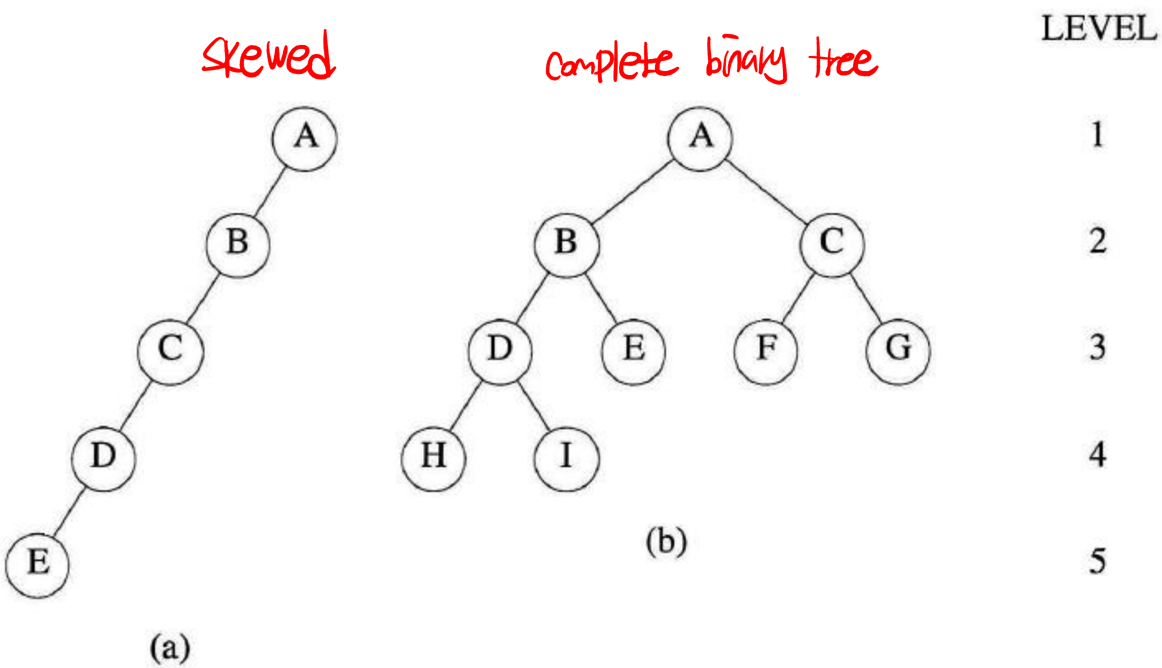
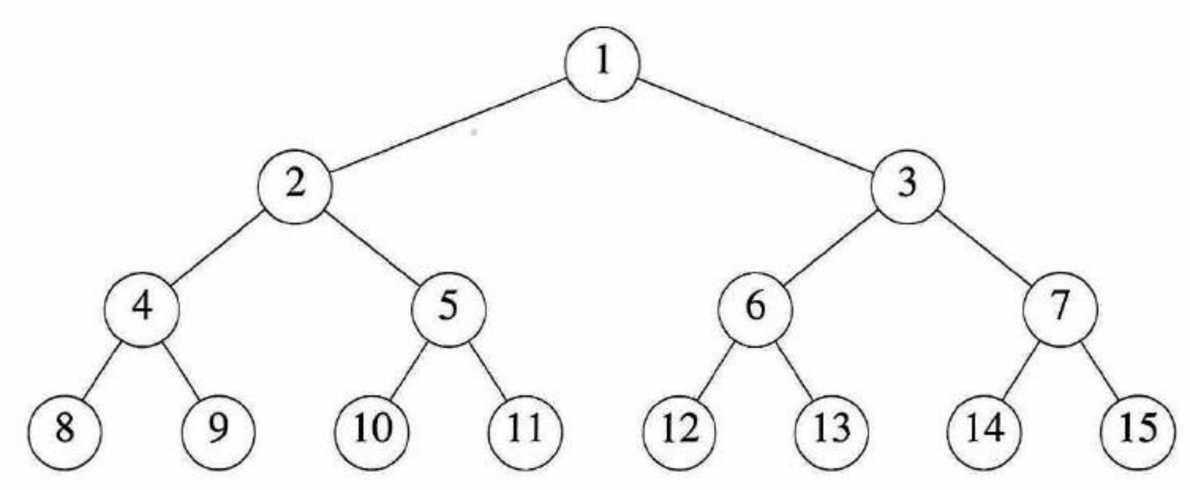


Figure 5.10: Skewed and complete binary trees

5.2.2 Properties of Binary Trees

- Max number of nodes
  - The max # of nodes on level i is 2^{i-1} $(i \geq 1)$
  - The max # of nodes in a BT of depth k is $2^k - 1$ $(k \geq 1)$

→ $$\sum_{i=1}^{k} (\text{maximum number of nodes on level } i) = \sum_{i=1}^{k} 2^{i-1} = 2^k - 1$$



1
2
3
4

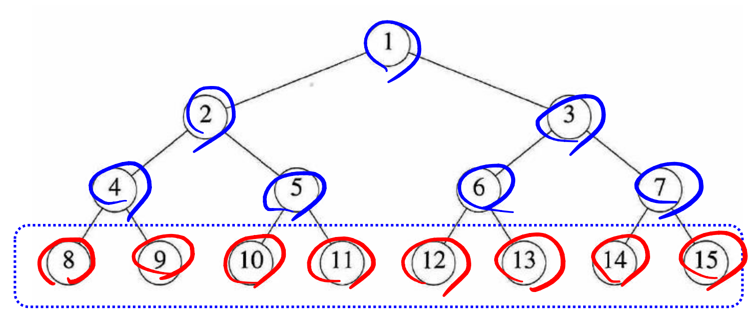
- Relation between # of leaf nodes and # of degree-2 nodes:

  $n_0 = n_2 + 1$

  – n_0 : # of leaf nodes 단말 노드=8

  – n_2 : # of nodes of degree 2

  degree가 2인 노드=7



[Proof]

  – n: total # of nodes

  – B: # of branches

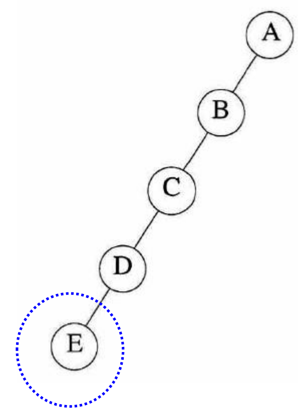
  – $n = n_0 + n_1 + n_2$

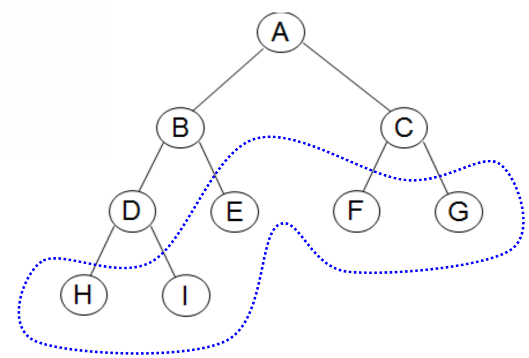
  – $n = B + 1$ B= n_1+2n_2

  $= n_1 + 2n_2 + 1$

  → $n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$

  → $n_0 = n_2 + 1$





포화

- **Full** binary tree
  - Every node other than the leaves has two children.
  - BT of depth k having 2^k-1 nodes, $k \geq 0$
  - Nodes: numbered from 1 to 2^k-1

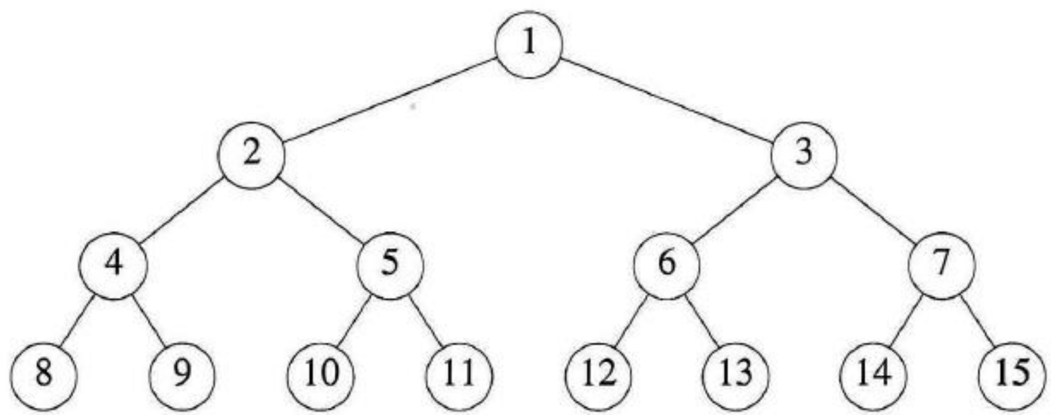


Figure 5.11: Full binary tree of depth 4 with sequential node numbers

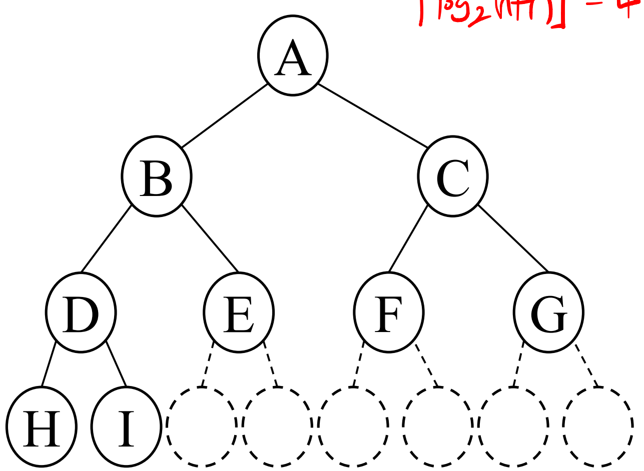
Full과 차이?

sol) Full은 각 level의 노드가 꽉 차있어야함

- **Complete** binary tree → 마지막 레벨을 제외하고 다른 레벨은 꽉차있음
  - Its nodes correspond to the nodes numbered from 1 to n in the full BT of depth k.
  - Height of a complete binary tree with n nodes: $\lceil \log_2(n+1) \rceil$ 올림

ex) $n=9$

$\lceil \log_2(n+1) \rceil = 4$



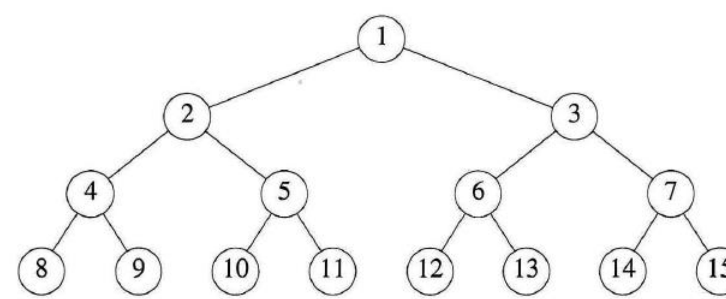
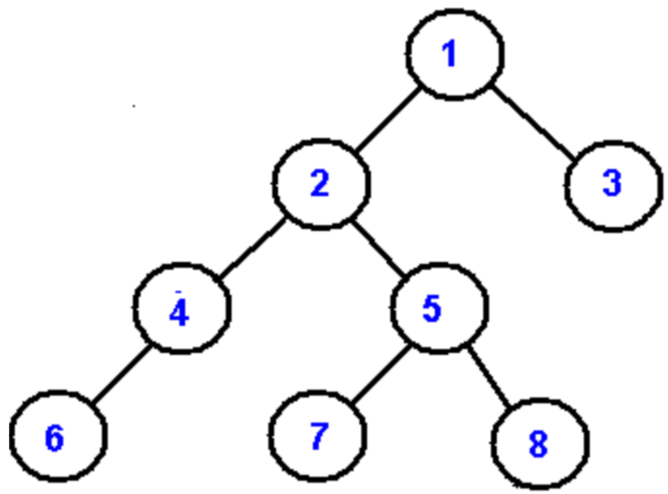


Figure 5.11: Full binary tree of depth 4 with sequential node numbers

binary tree (o)

Complete 〃 (x)

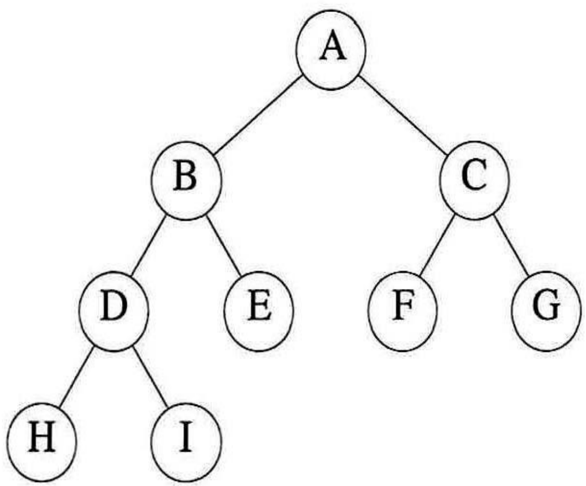
Full (x)



5.2.3 Binary Tree Representations

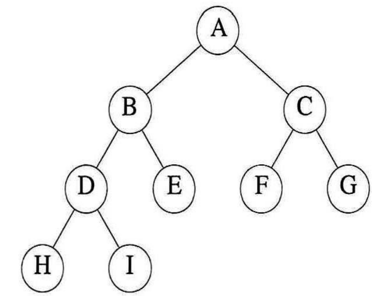
1) Array Representation

완전 이진 트리일때는 Array가 좋음



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

(b) Tree of Figure 5.10(b)

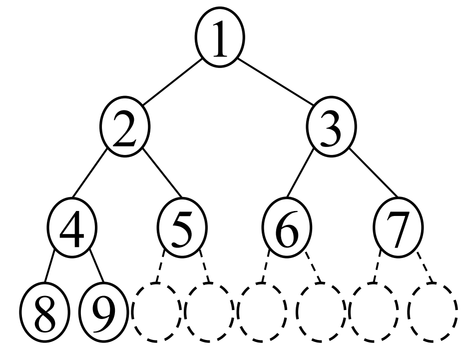


- Case : Complete BT with $\overset{9}{n}$ nodes

(1) parent(i): $\lfloor i/2 \rfloor$ 내림 if $i \neq 1$

(2) LeftChild(i): $2i$ if $2i \leq n$
no left child if $2i > n$

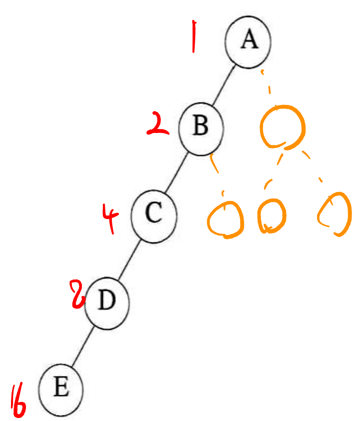
(3) RightChild(i) $2i+1$ if $2i+1 \leq n$
no left child if $2i+1 > n$



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

(b) Tree of Figure 5.10(b)

- Disadvantages skewed tree는 array로 표현하면 memory wasted
  - Waste space;
  - A skewed tree of depth k will require 2^k-1 spaces (only k will be used)
  - Insertion or deletion of nodes : …



	tree
[0]	–
[1]	A
[2]	B
[3]	–
[4]	C
[5]	–
[6]	–
[7]	–
[8]	D
[9]	–
.	.
.	.
.	.
[16]	E

2) Linked Representation

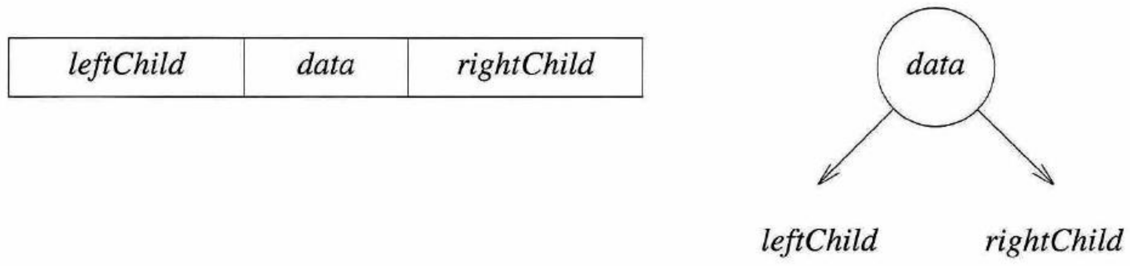


Figure 5.13: Node representations

```
typedef struct node *treePointer;
typedef struct node{
        int data;
        treePointer leftChild, rightChild;
        };
```

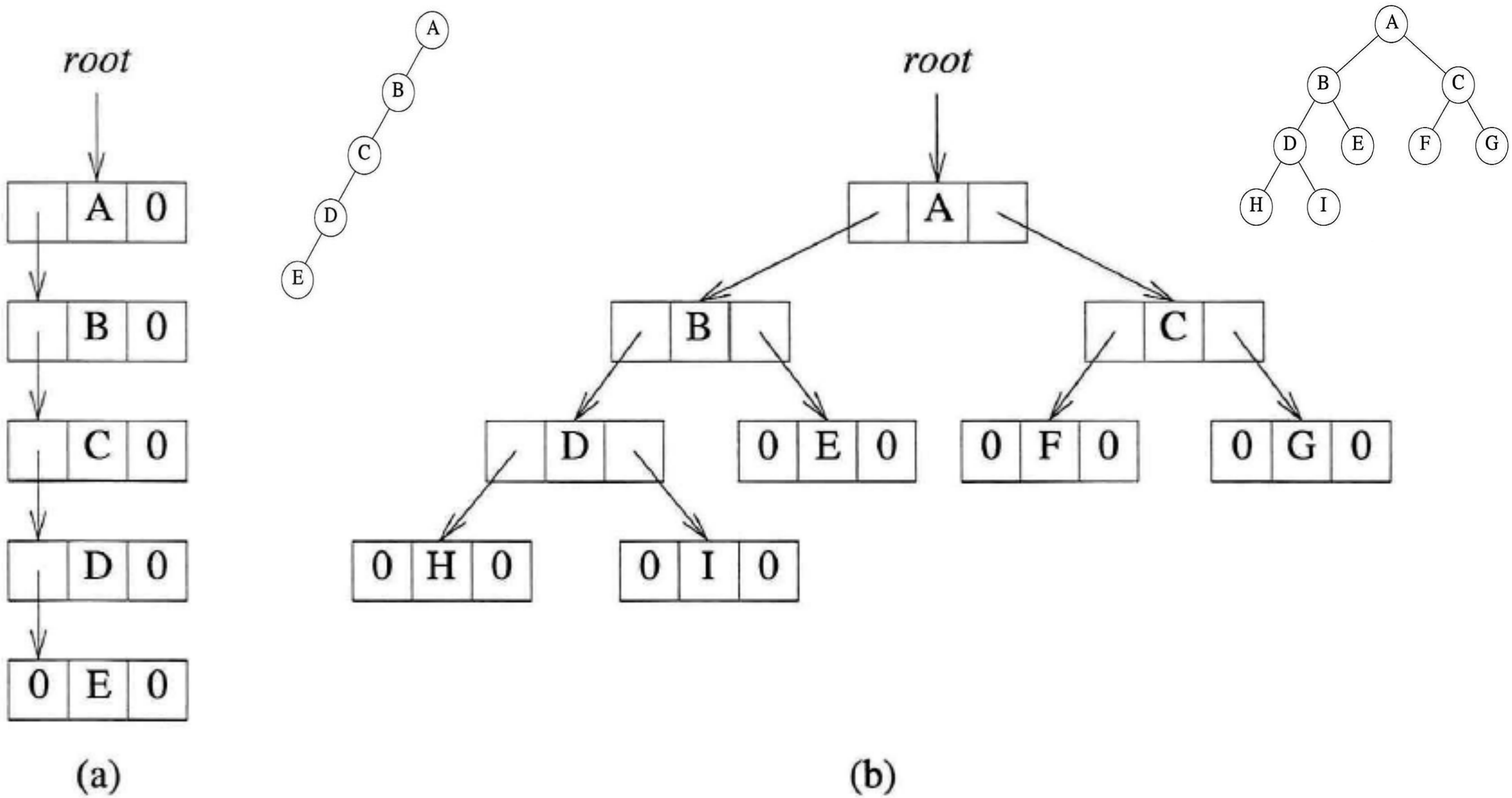


Figure 5.14: Linked representation for the binary trees of Figure 5.10

5.1 Introduction

5.2 Binary Trees

5.3 Binary Trees Traversals

5.4 Additional Binary Tree Operations

5.5 Threaded Binary Trees

5.6 Heaps

5.7 Binary Search Trees