Unit 1: Introduction and basic concepts (Chap. 1, 2)

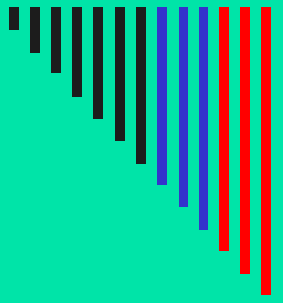
Unit 2: Linear lists (Stack, Queues, General linear lists)

Unit 3: Non-Linear Lists

- Tree (Chap. 6: Intro. to trees)
- Two-way Tree
  - Chap. 7: Binary search trees
  - Chap. 8: AVL trees
  - Chap. 9: Heap
- Multiway Trees (Chap.10)
- Graph (Chap. 11)

Unit 4: Sorting (Chap. 12)

Unit 5: Searching (Chap. 13)



Chapter 6

Introduction to Trees

Objectives

Upon completion you will be able to:

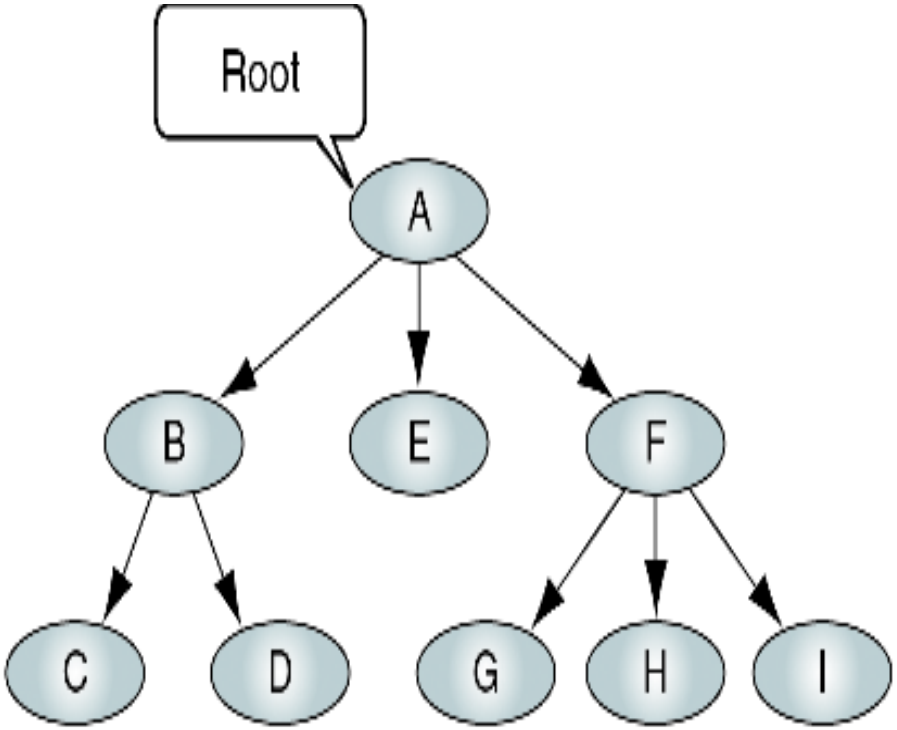
- **Understand and use basic tree terminology and concepts**
- **Recognize and define the basic attributes of a binary tree**
- **Process trees using depth-first and breadth-first traversals**
- **Parse expressions using a binary tree**
- **Design and implement Huffman trees**
- **Understand the basic use and processing of general trees**

6-1 Basic Tree Concepts

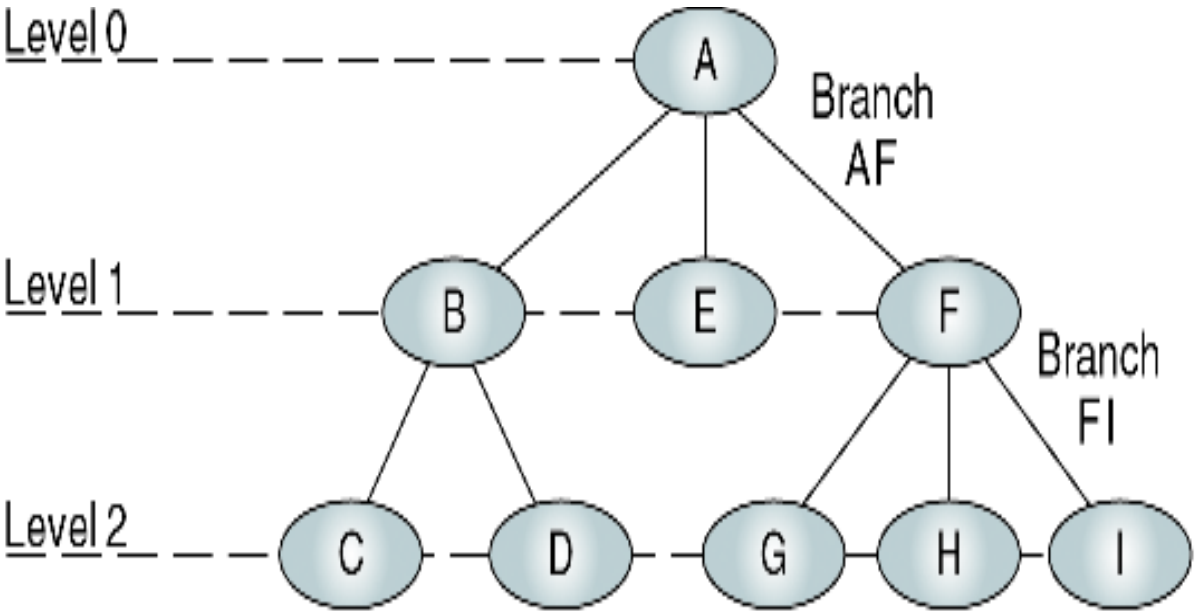
We begin the Chapter with a discussion of the terminology used with trees. Once the terminology is understood, we discuss three user-oriented tree representations: general trees, indented parts lists, and parenthetical trees.

- Terminology
- User Representation

Tree



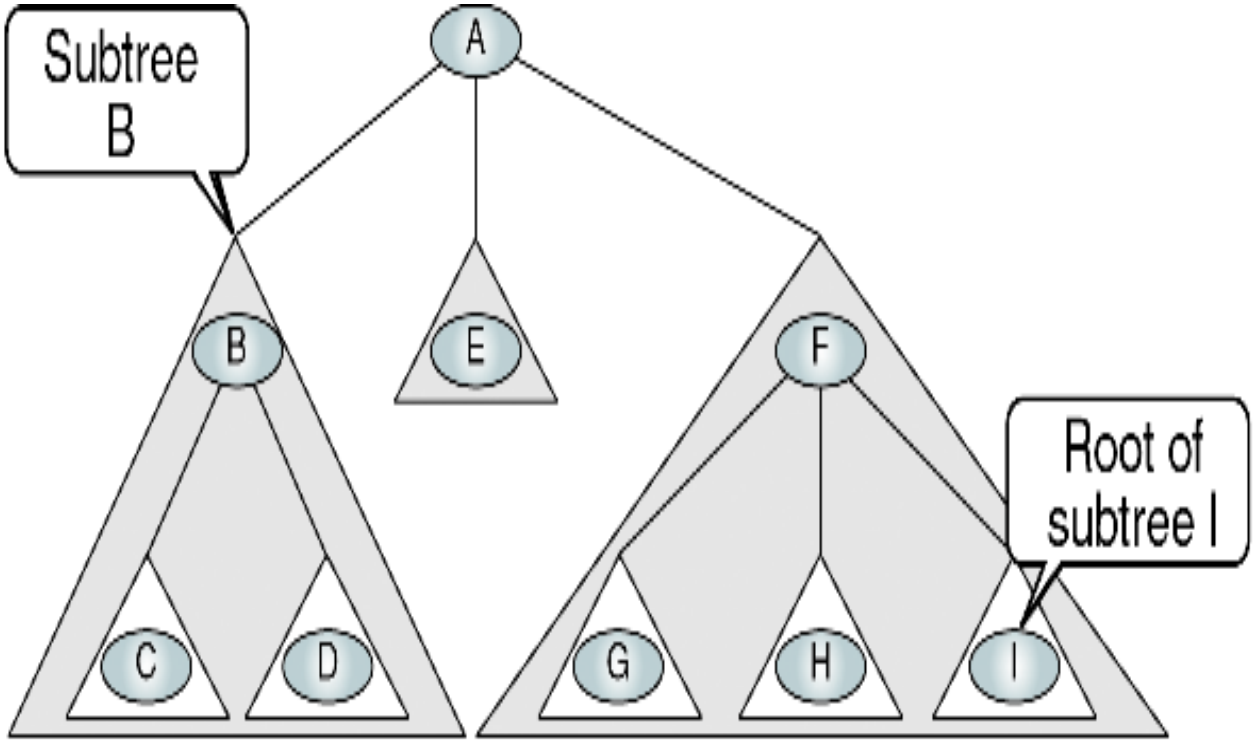
Tree Nomenclature 命名法



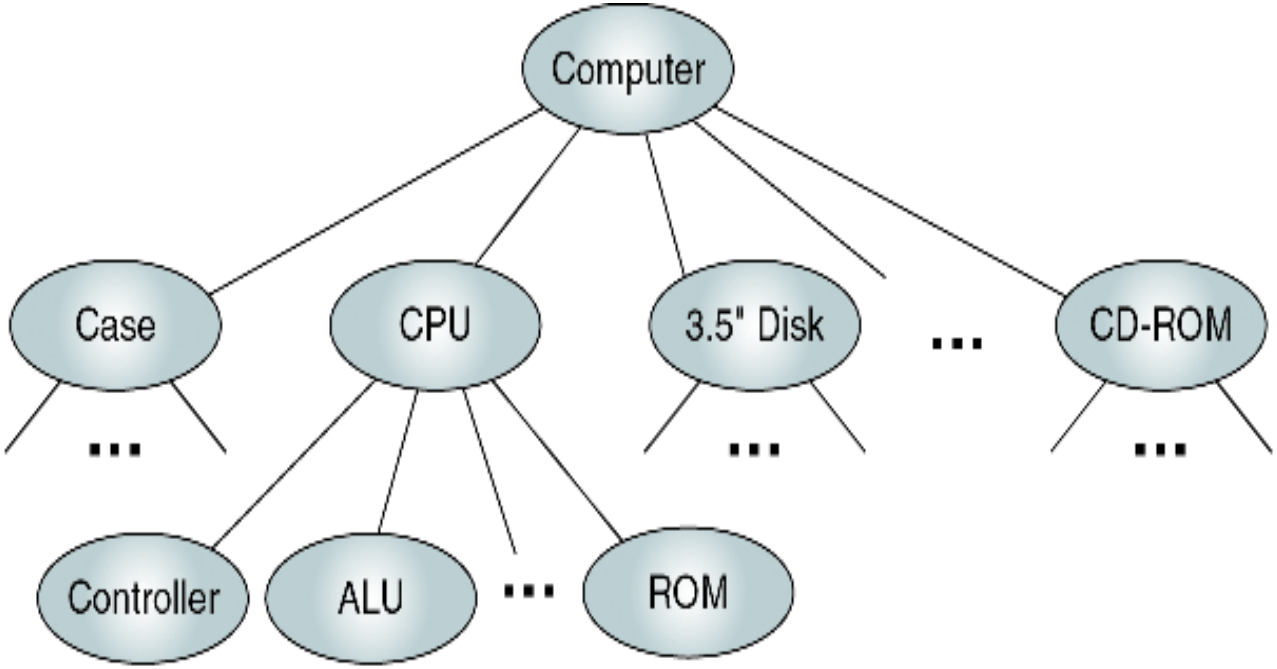
Height (depth) = 3

Root: A	Siblings: {B,E,F}, {C,D}, {G,H,I}
Parents: A, B, F	Leaves: C,D,E,G,H,I
Children: B, E, F, C, D, G, H, I	Internal nodes: B,F

Subtrees



Computer Parts List as a General Tree

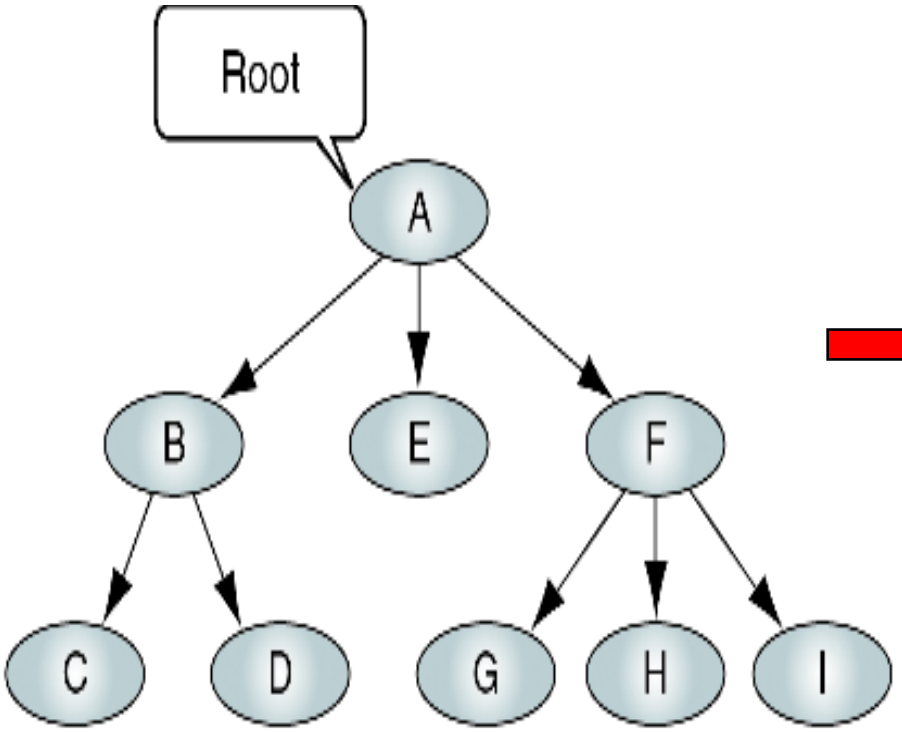


Computer Bill of Materials

Part number	Description
301	Computer
301-1	Case
...	...
301-2	CPU
301-2-1	Controller
301-2-2	ALU
...	...
301-2-9	ROM
301-3	3.5" Disk
...	...
301-9	CD-ROM
...	...

Convert General Tree to Parenthetical Notation

- Open parenthesis indicates the start of of a new level
- Closeing parenthesis completes the current level and move up one level in the tree



traverse的方式不只一種

A (B (C D) E F (G H I))

Convert General Tree to Parenthetical Notation

```
Algorithm ConvertToParen (root, output)
Convert a general tree to parenthetical notation.
   Pre  root is a pointer to a tree node
   Post output contains parenthetical notation
1  Place root in output
2  if (root is a parent)
   1  Place an open parenthesis in the output
   2  ConvertToParen (root's first child)
   3  loop (more siblings)
      1  ConvertToParen (root's next child)
   4  end loop
   5  Place close parenthesis in the output
3  end if
4  return
end ConvertToParen
```

6-2 Binary Trees

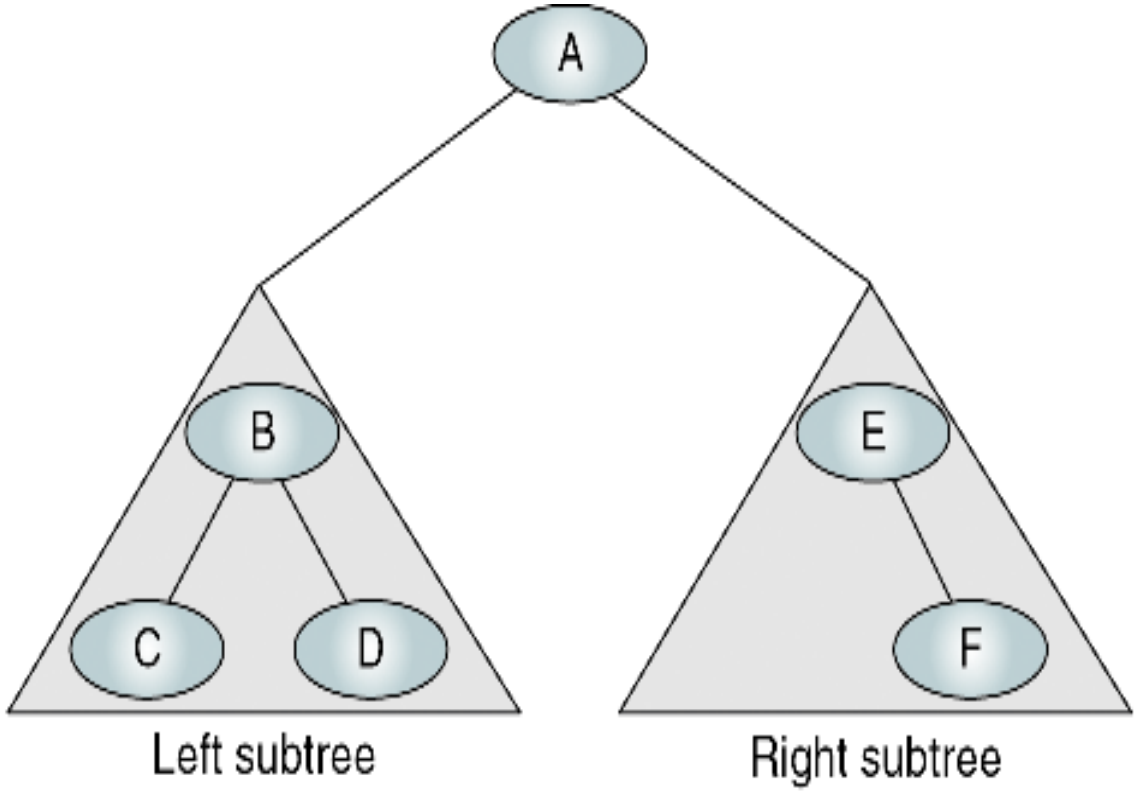
A binary tree can have no more than two descendents. In this section we discuss the properties of binary trees, four different binary tree traversals, and two applications, expression trees and Huffman Code.

- Properties
- Binary Tree Traversals
- Expression Trees
- Huffman Code

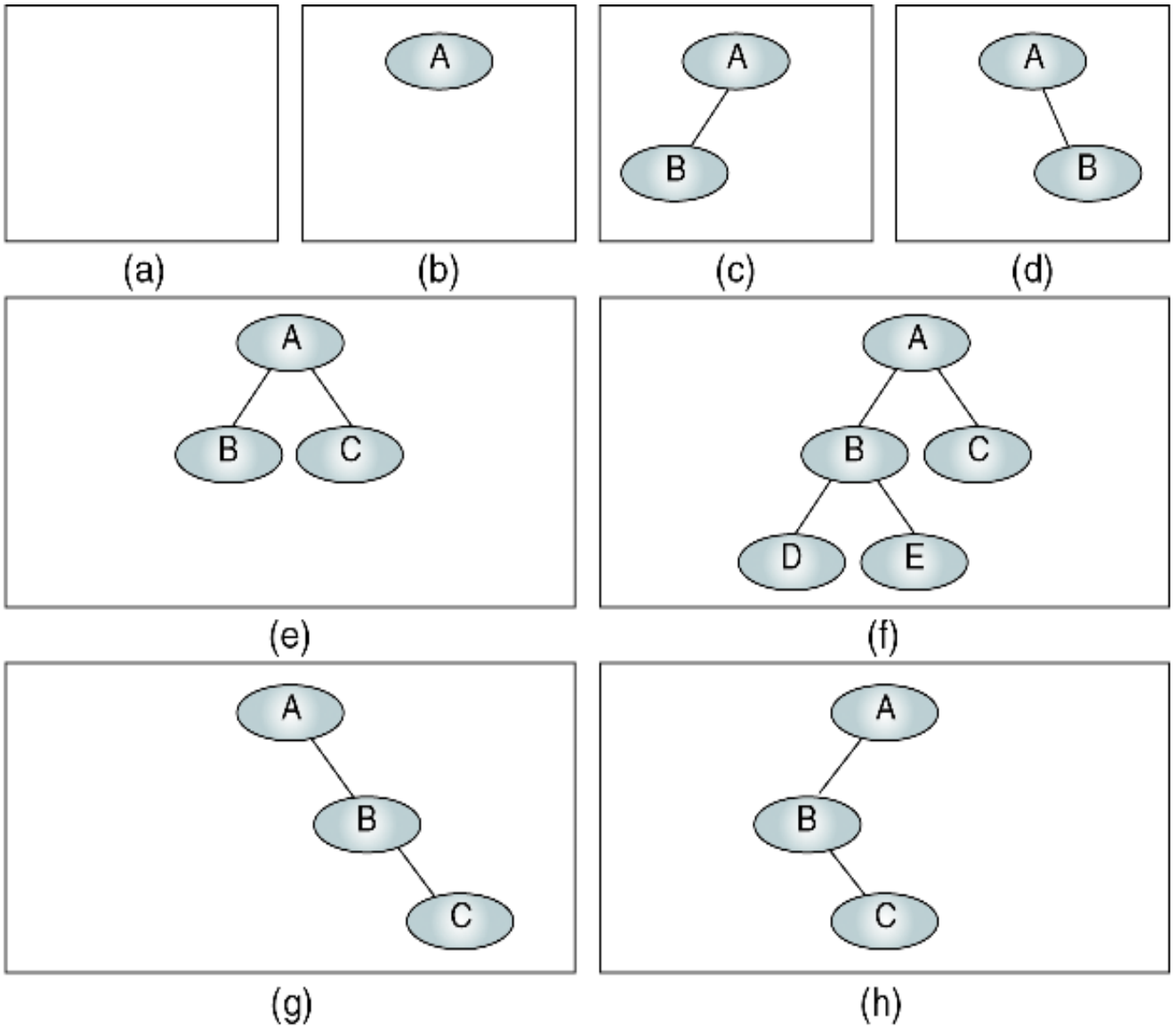
Binary Tree

- A node in a binary tree can have **no more than two subtree**

每個subtree都要符合條件



Collection of Binary Trees



Properties of Binary Tree

Given *N* nodes in a binary tree

Maximum height of binary tree:

$$H_{\max} = N$$

Minimum height of binary tree:

$$H_{\min} = \lfloor \log_2 N \rfloor + 1$$

跟search的O(n)有關

Given height of the binary tree *H*

Minimum number of nodes in the tree:

$$N_{\min} = H$$

Maximum number of nodes in the tree:

$$N_{\max} = 2^H - 1$$

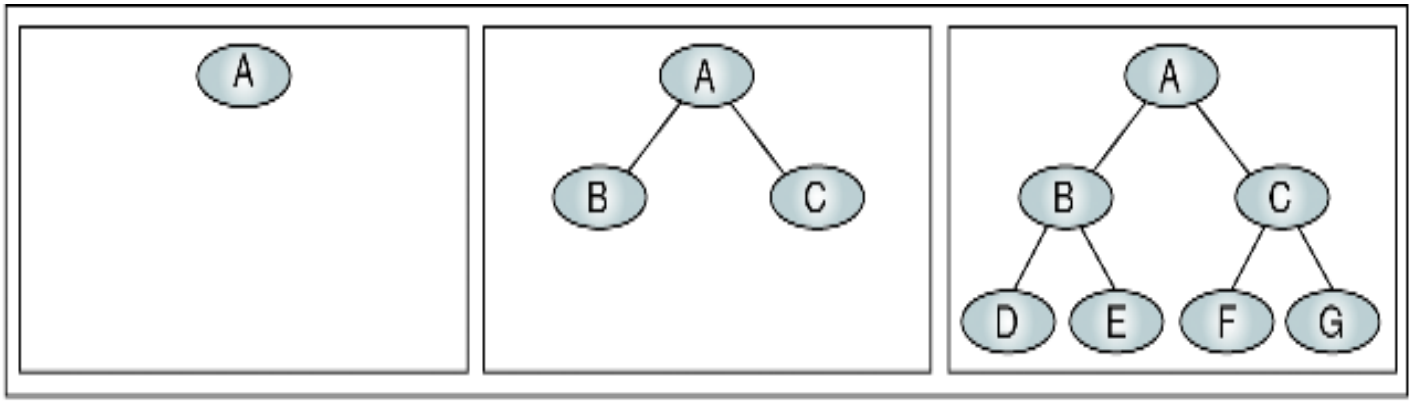
Given height of the left subtree H_L and right subtree H_R

Balance factor:

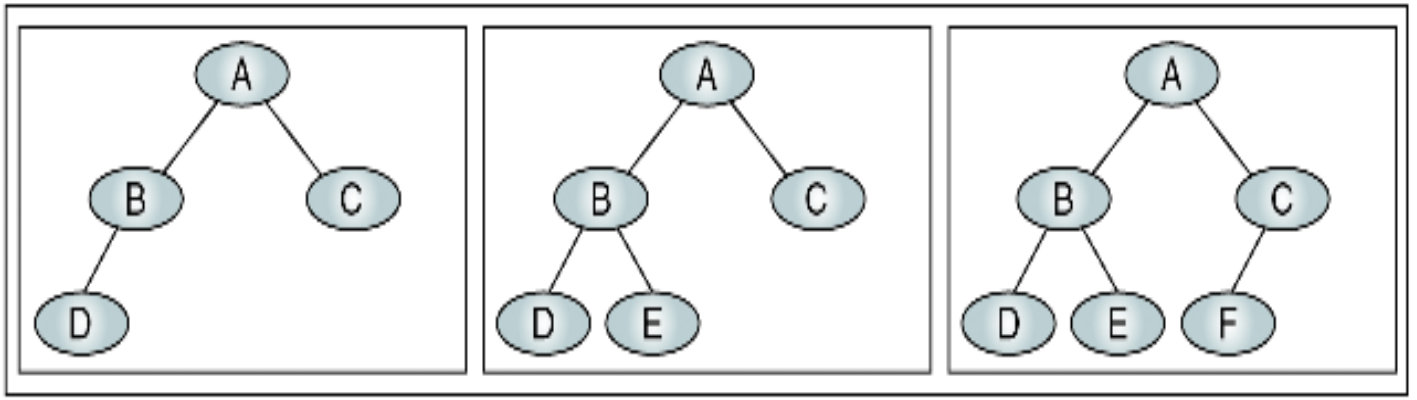
$$B = H_L - H_R$$

Note: Balanced binary tree *B* = -1, 0, or +1
e.g., AVL tree

Complete and Nearly Complete Trees



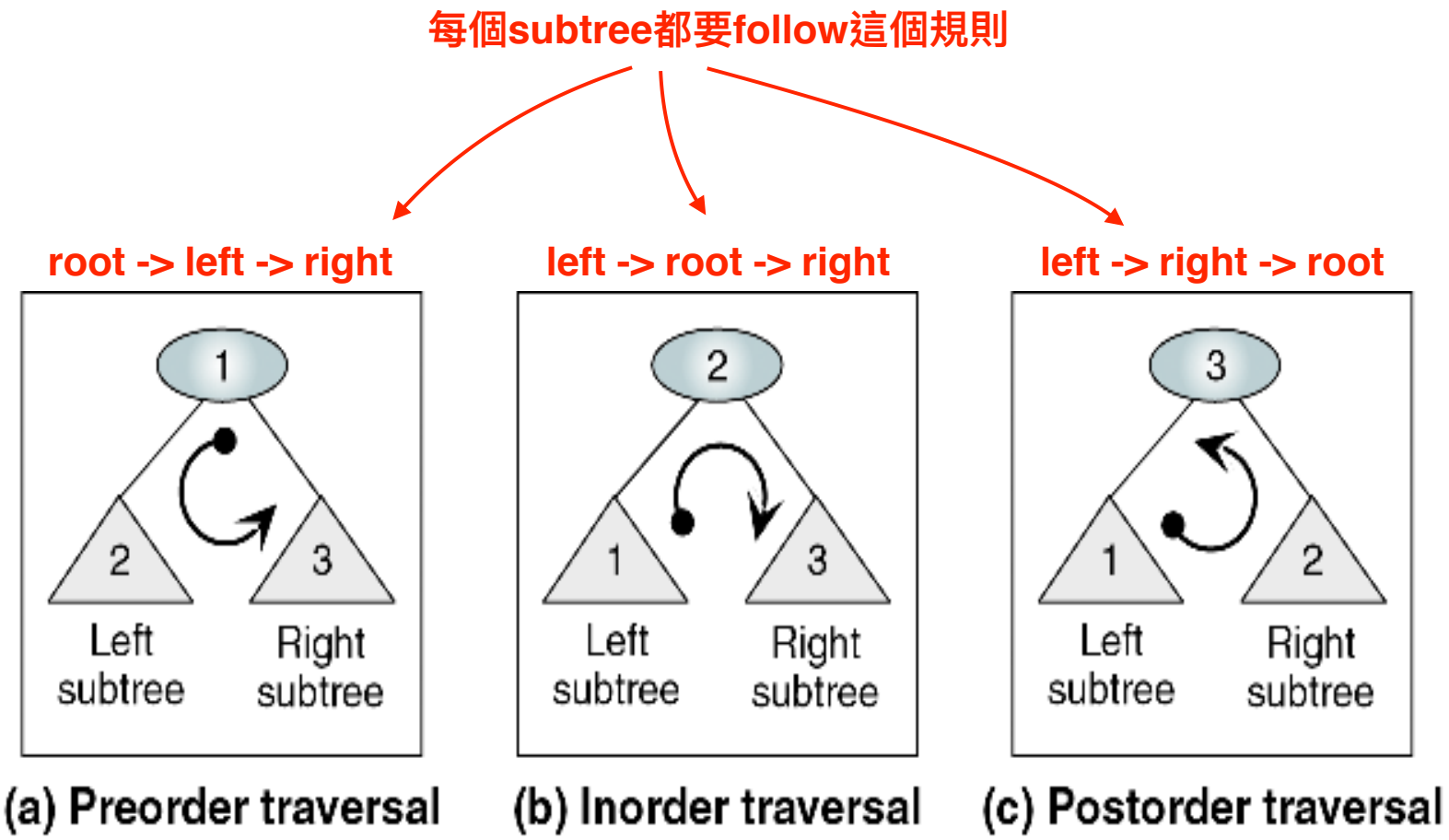
(a) Complete trees (at levels 0, 1, and 2)



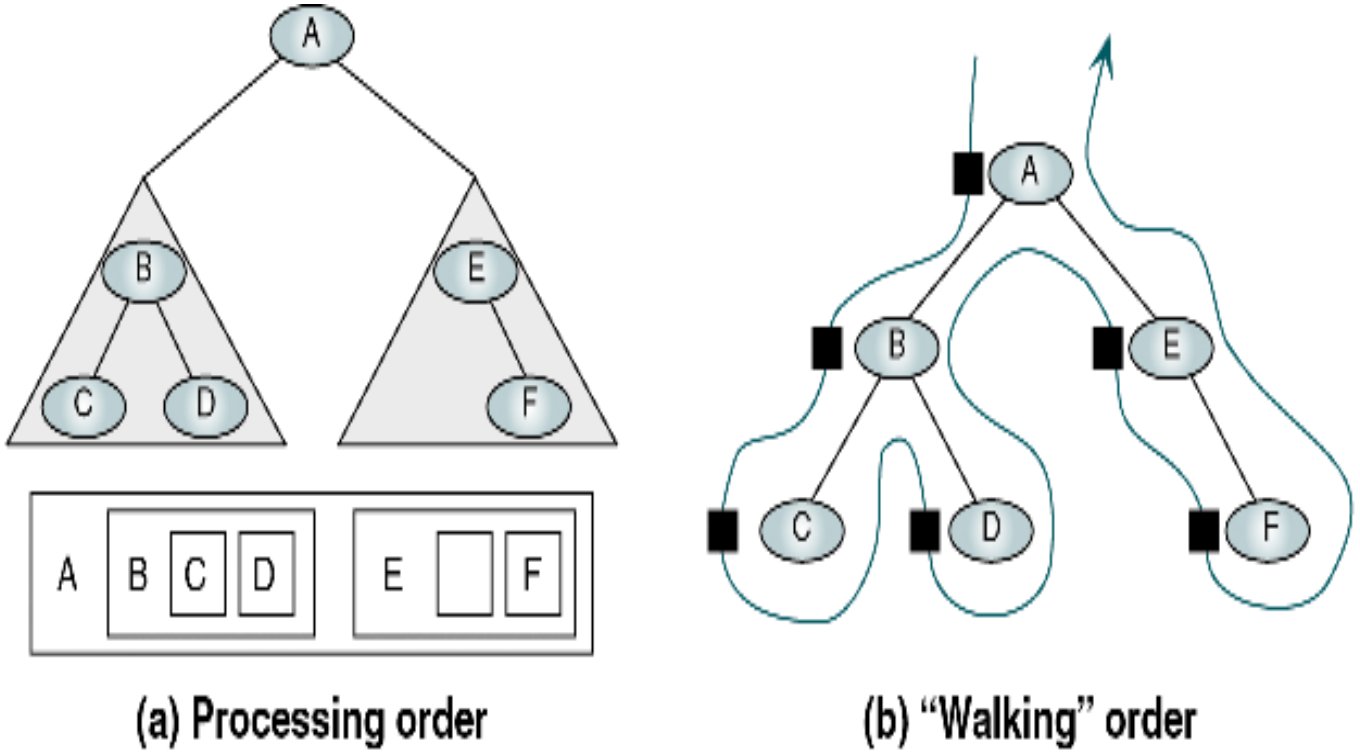
(b) Nearly complete trees (at level 2)

缺的一定在右邊

Binary Tree Traversals- Depth-first Traversals



Preorder Traversal -- ABCDEF



(a) Processing order

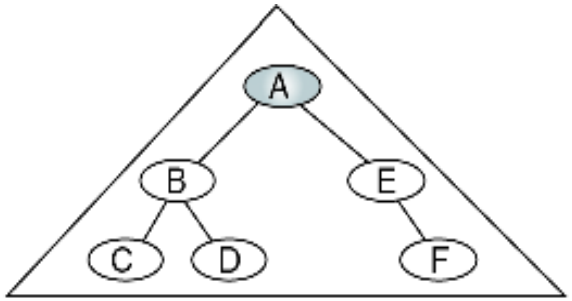
(b) "Walking" order

Preorder Traversal of a Binary Tree

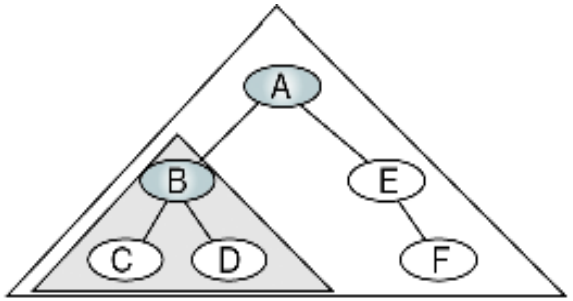
root -> left -> right

```
Algorithm preOrder (root)
Traverse a binary tree in node-left-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   process (root)
   2   preOrder (leftSubtree)
   3   preOrder (rightSubtree)
2  end if
end preOrder
```

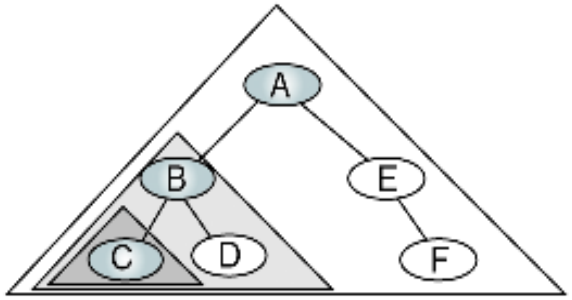
Algorithmic Traversal of Binary Tree



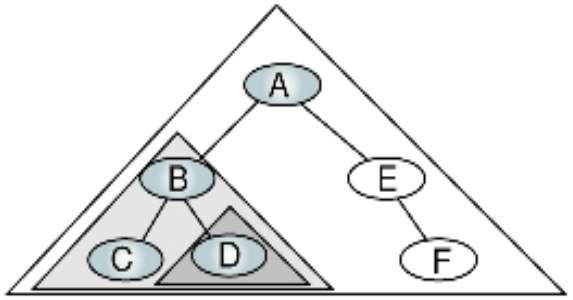
(a) Process tree A



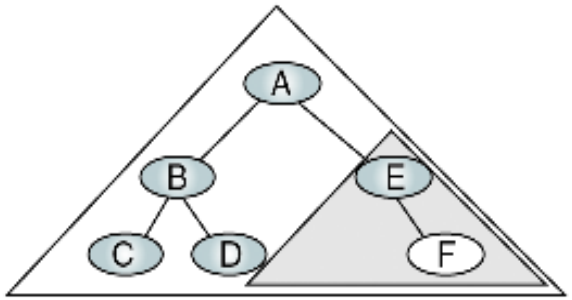
(b) Process tree B



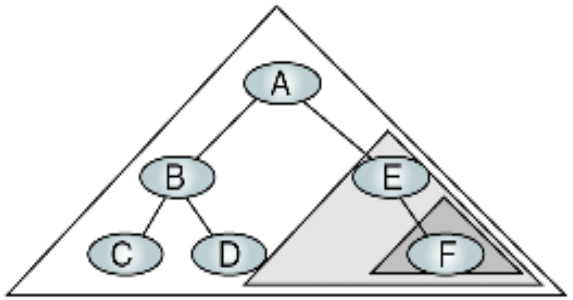
(c) Process tree C



(d) Process tree D

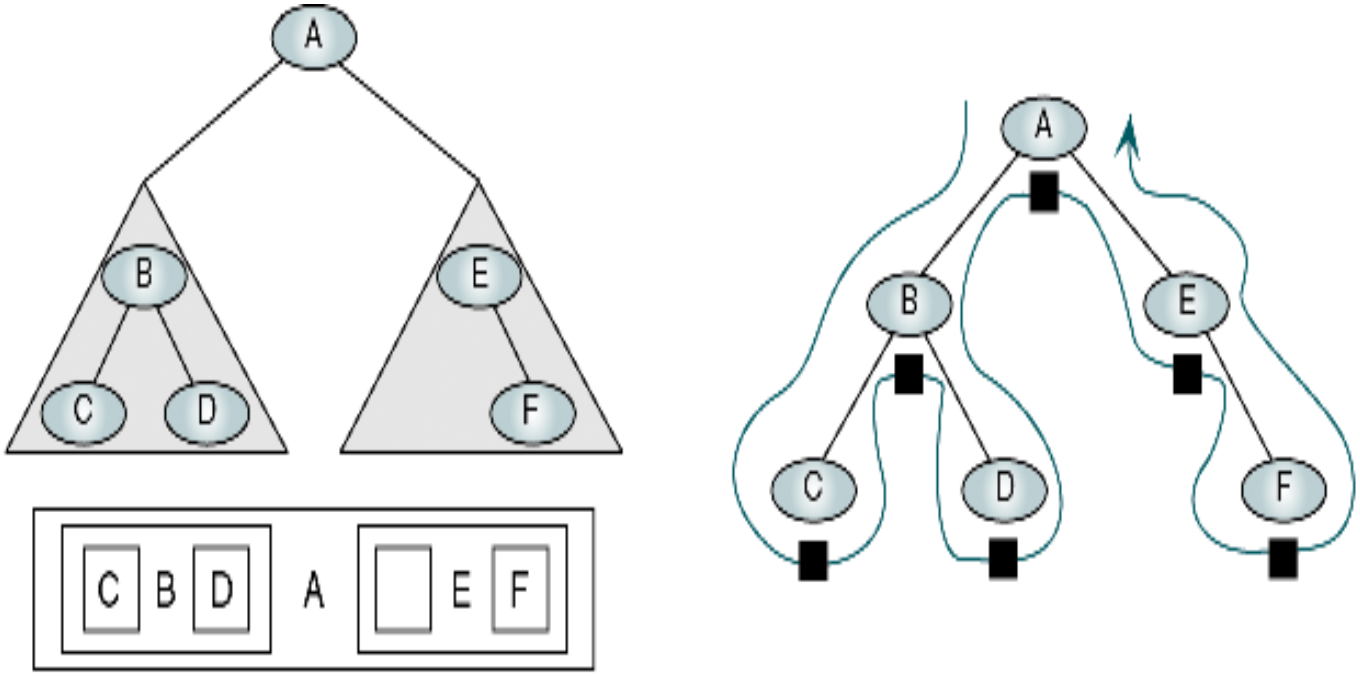


(e) Process tree E



(f) Process tree F

Inorder Traversal -- CBDAEF



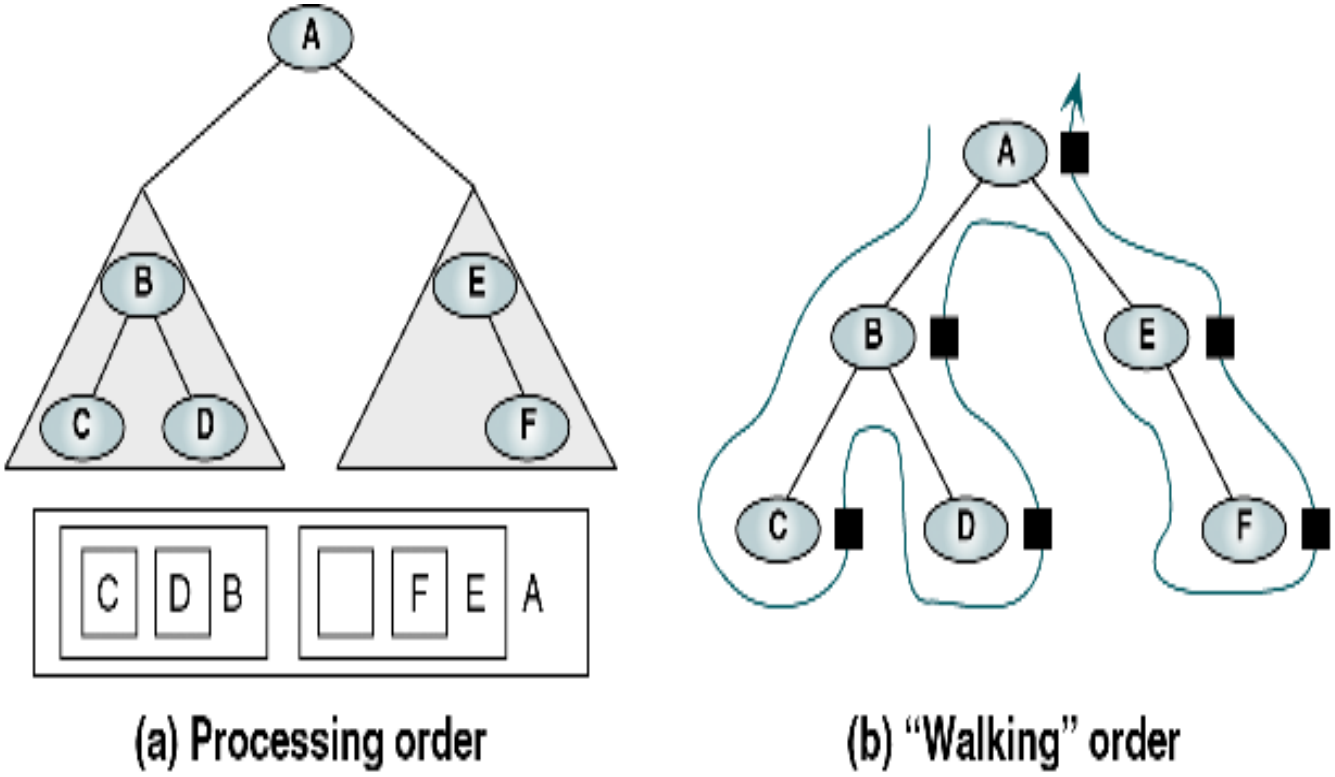
(a) Processing order

(b) "Walking" order

Inorder Traversal of Binary Tree

```
Algorithm inOrder (root)
Traverse a binary tree in left-node-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   inOrder (leftSubTree)
   2   process (root)
   3   inOrder (rightSubTree)
2  end if
end inOrder
```

Postorder Traversal -- CDBFEA

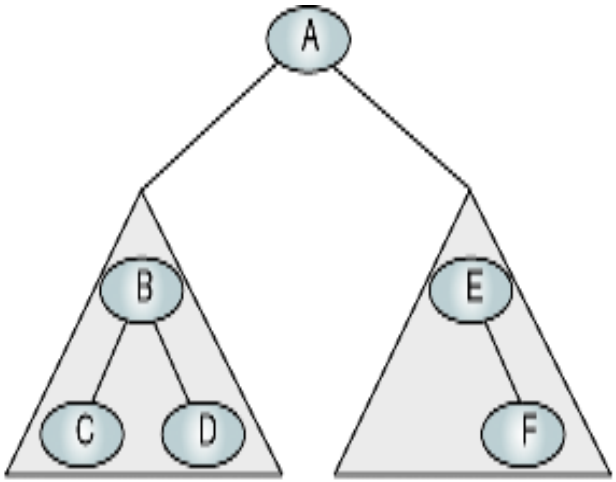


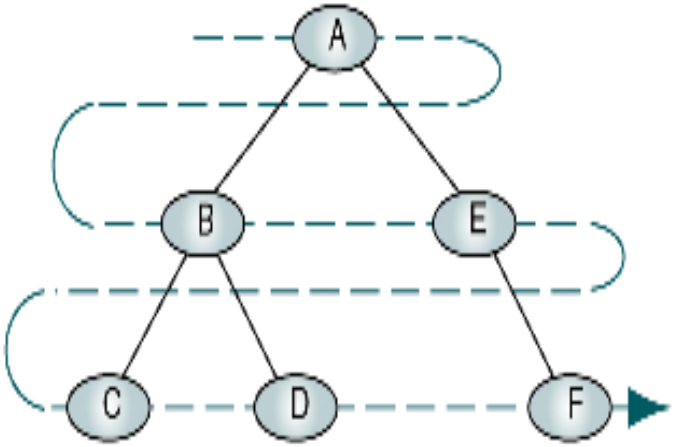
Postorder Traversal of Binary Tree

```
Algorithm postOrder (root)
Traverse a binary tree in left-right-node sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   postOrder (left subtree)
   2   postOrder (right subtree)
   3   process (root)
2  end if
end postOrder
```

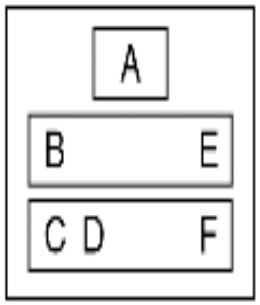
Breadth-first Traversal

- A queue (FIFO) is used to traverse a tree in breadth-first order
- A stack (FILO) is used to trverse a tree in depth-first order (why ?)





(b) "Walking" order



(a) Processing order

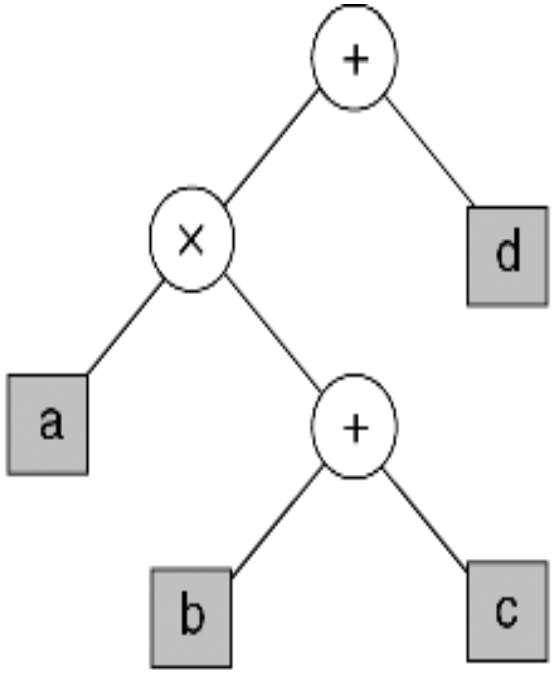
Breadth-first Tree Traversal

```
Algorithm breadthFirst (root)
Process tree using breadth-first traversal.
   Pre    root is node to be processed
   Post   tree has been processed
 1 set currentNode to root
 2 createQueue (bfQueue)
 3 loop (currentNode not null)
   1   process (currentNode)
   2   if (left subtree not null)
       1   enqueue (bfQueue, left subtree)
   3   end if
   4   if (right subtree not null)
       1   enqueue (bfQueue, right subtree)
   5   end if
   6   if (not emptyQueue(bfQueue))
       1   set currentNode to dequeue (bfQueue)
   7   else
       1   set currentNode to null
   8   end if
 4 end loop
 5 destroyQueue (bfQueue)
end breadthFirst
```

Infix Expression and Expression Tree

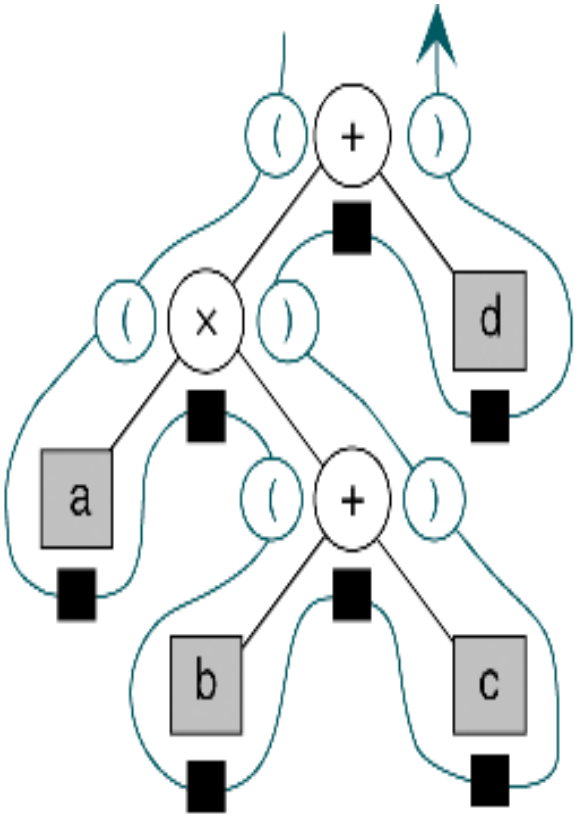
- Each leaf is an operand
- The root and internal nodes are operators
- Subtrees are subexpressions, with the root being an operator

a × (b + c) + d



Infix Traversal of Expression Tree

((a x (b + c)) + d)



Infix Expression Tree Traversal

```
Algorithm infix (tree)
Print the infix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the infix expression has been printed
1  if (tree not empty)
   1   if (tree token is an operand)
       1   print (tree-token)
   2   else
       1   print (open parenthesis)
       2   infix (tree left subtree)
       3   print (tree token)
       4   infix (tree right subtree)
       5   print (close parenthesis)
   3   end if
2  end if
end infix
```

Postfix Traversal of Expression Tree

```
Algorithm postfix (tree)
Print the postfix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the postfix expression has been printed
1  if (tree not empty)
   1  postfix (tree left subtree)
   2  postfix (tree right subtree)
   3  print   (tree token)
2  end if
end postfix
```

Prefix Traversal of Expression Tree

```
Algorithm prefix (tree)
Print the prefix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the prefix expression has been printed
1  if (tree not empty)
   1   print  (tree token)
   2   prefix (tree left subtree)
   3   prefix (tree right subtree)
2  end if
end prefix
```

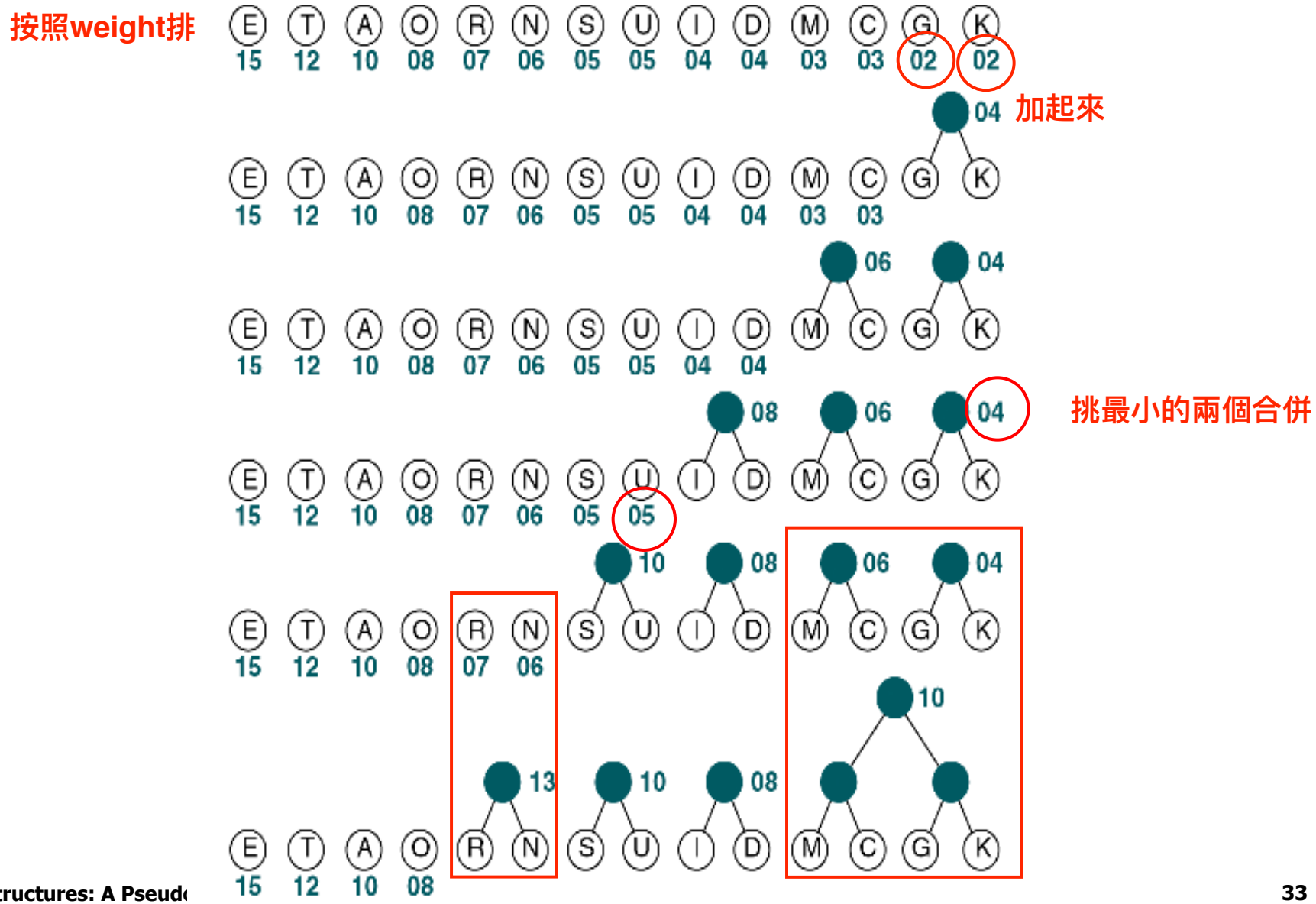
Huffman Coding

- Hufman code assign shorter codes to characters that accur more frequently and longer codes to those that occur less frequently
- Huffman code is a populer data compression algorithm

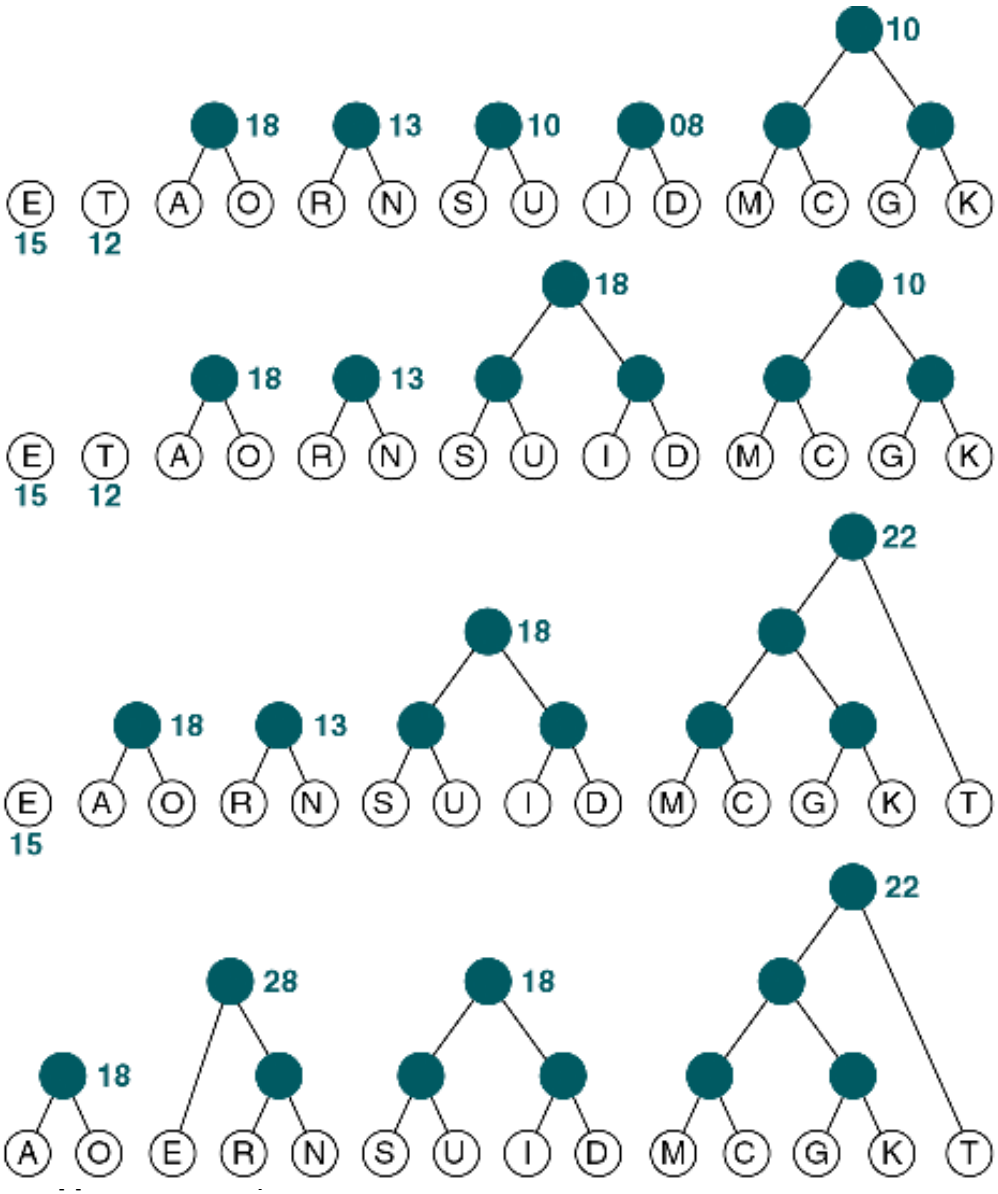
可以想成出現的比率

Character	Weight	Character	Weight	Character	Weight
A	10	I	4	R	7
C	3	K	2	S	5
D	4	M	3	T	12
E	15	N	6	U	5
G	2	O	8		

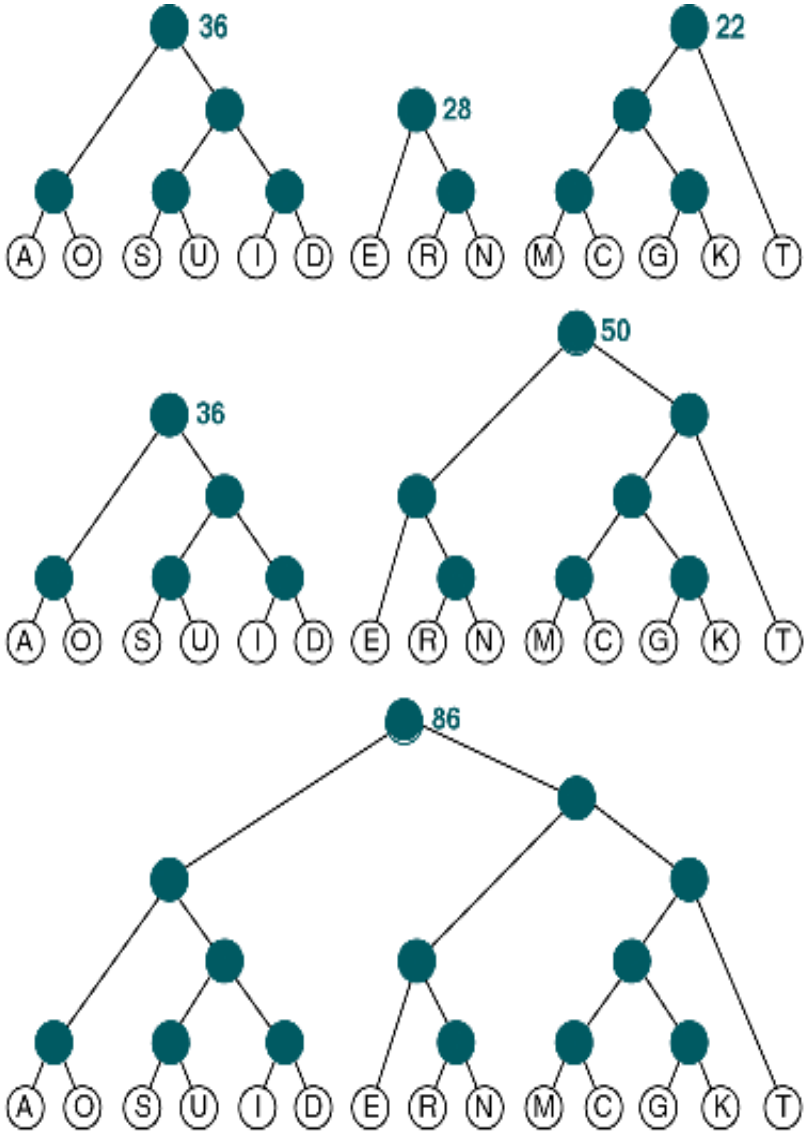
Huffman Tree – Part I



Huffman Tree – Part II



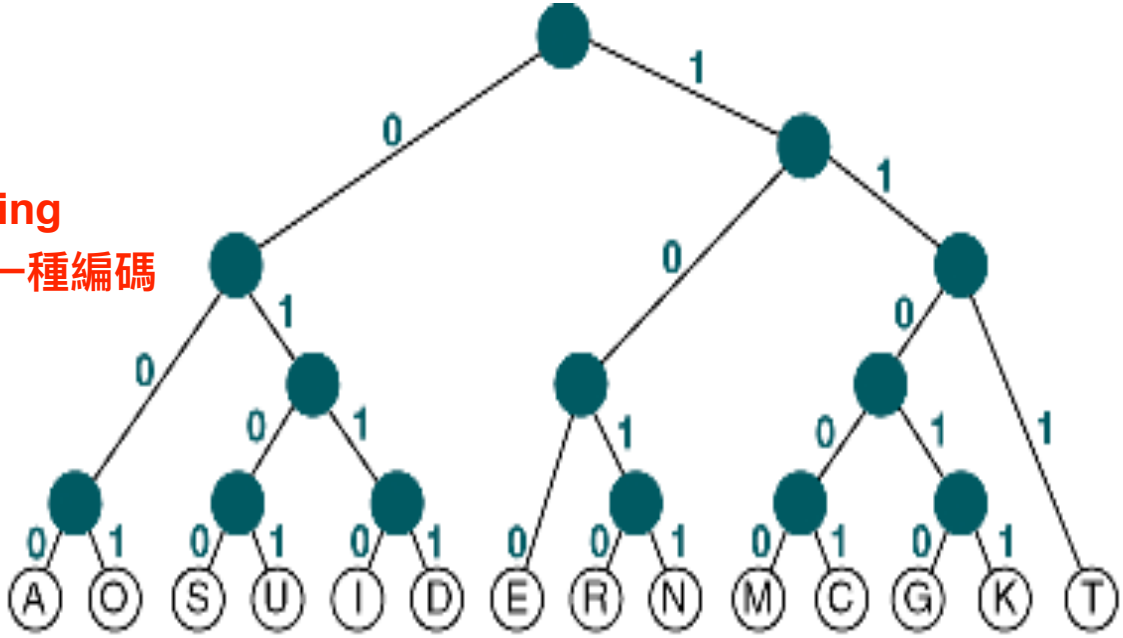
Huffman Tree – Part III



Huffman Code Assignment

one-to-one mapping

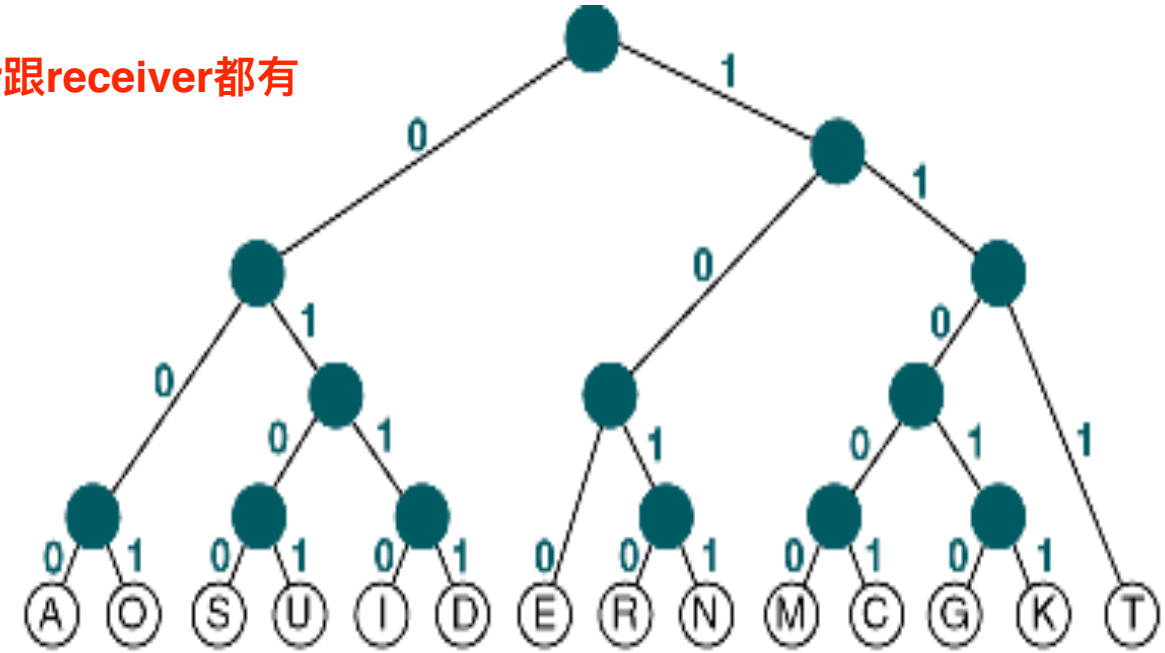
每個字母「只」有一種編碼



A = 000	U = 0101	E = 100	M = 11000	K = 11011
O = 001	I = 0110	R = 1010	C = 11001	T = 111
S = 0100	D = 0111	N = 1011	G = 11010	

Decoding of Huffman Code

這個tree sender跟receiver都有



Sender: 000110100010010111110000010101101100111

Receiver: AGOODMARKET

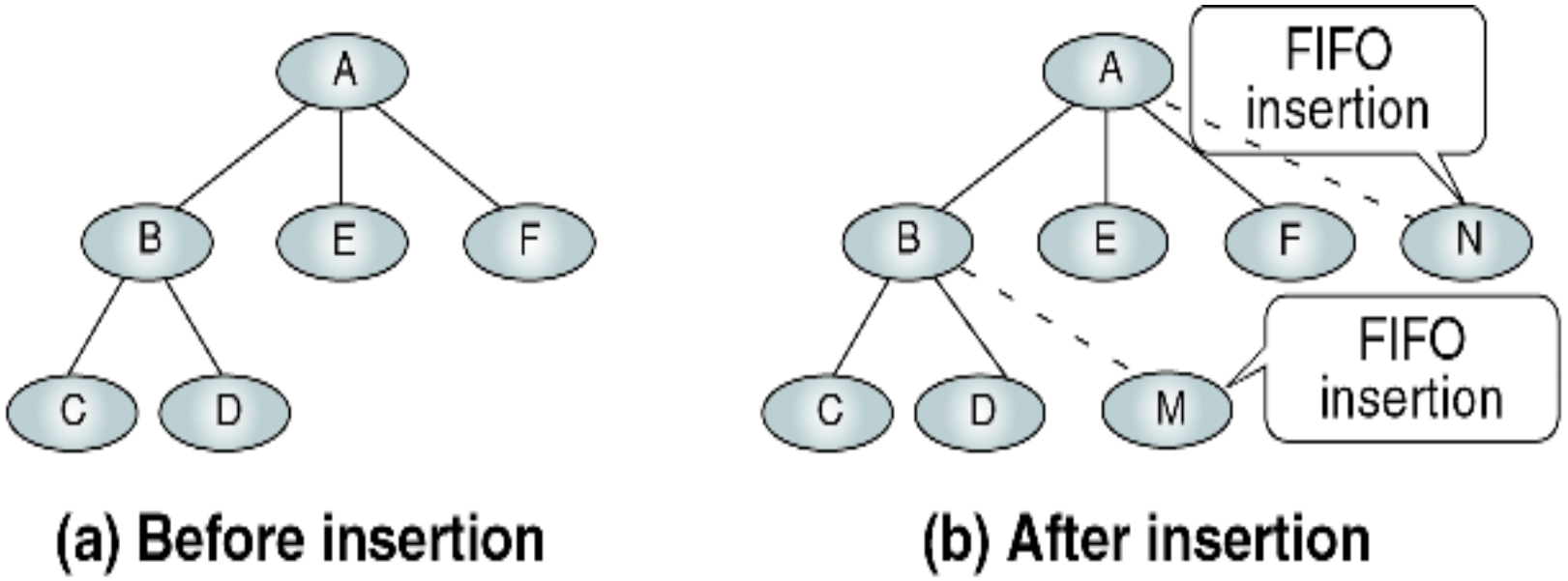
6-3 General Trees

A general tree can have an unlimited number of descendents. We discuss three topics in this section: general tree insertions, deletions, and converting a general tree to a binary tree.

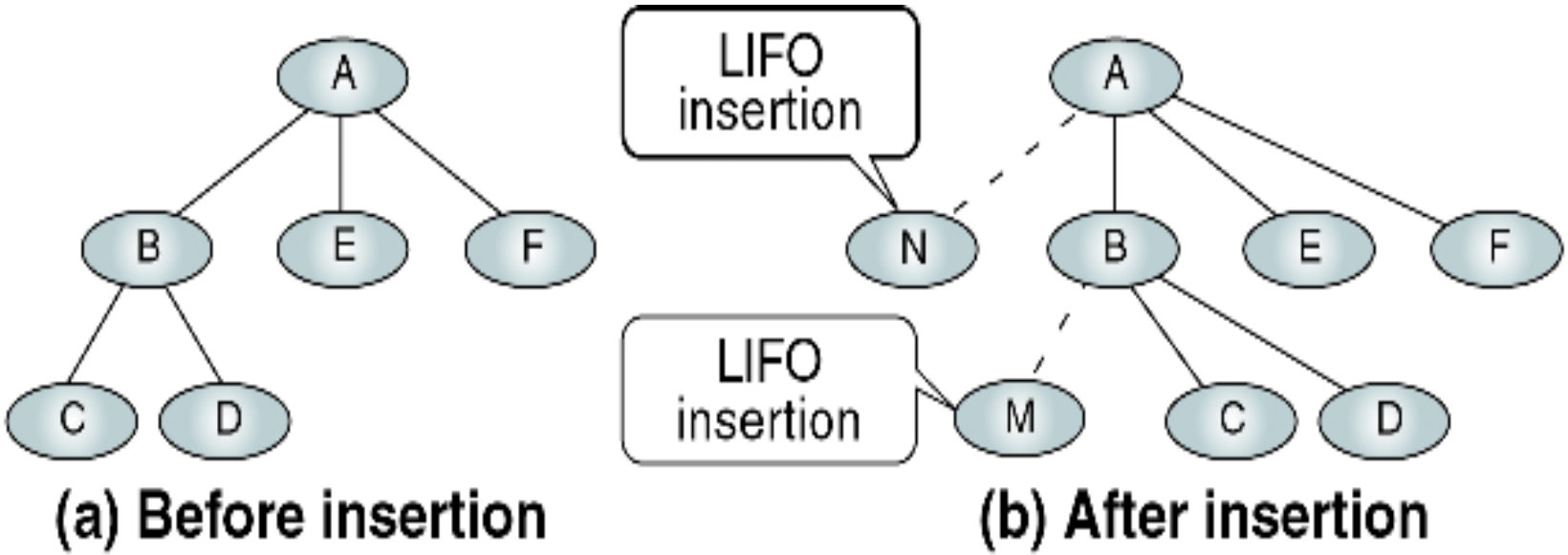
- Insertions into General Trees
- General Tree Deletions
- Changing a General Tree to a Binary Tree

FIFO Insertion into General Trees

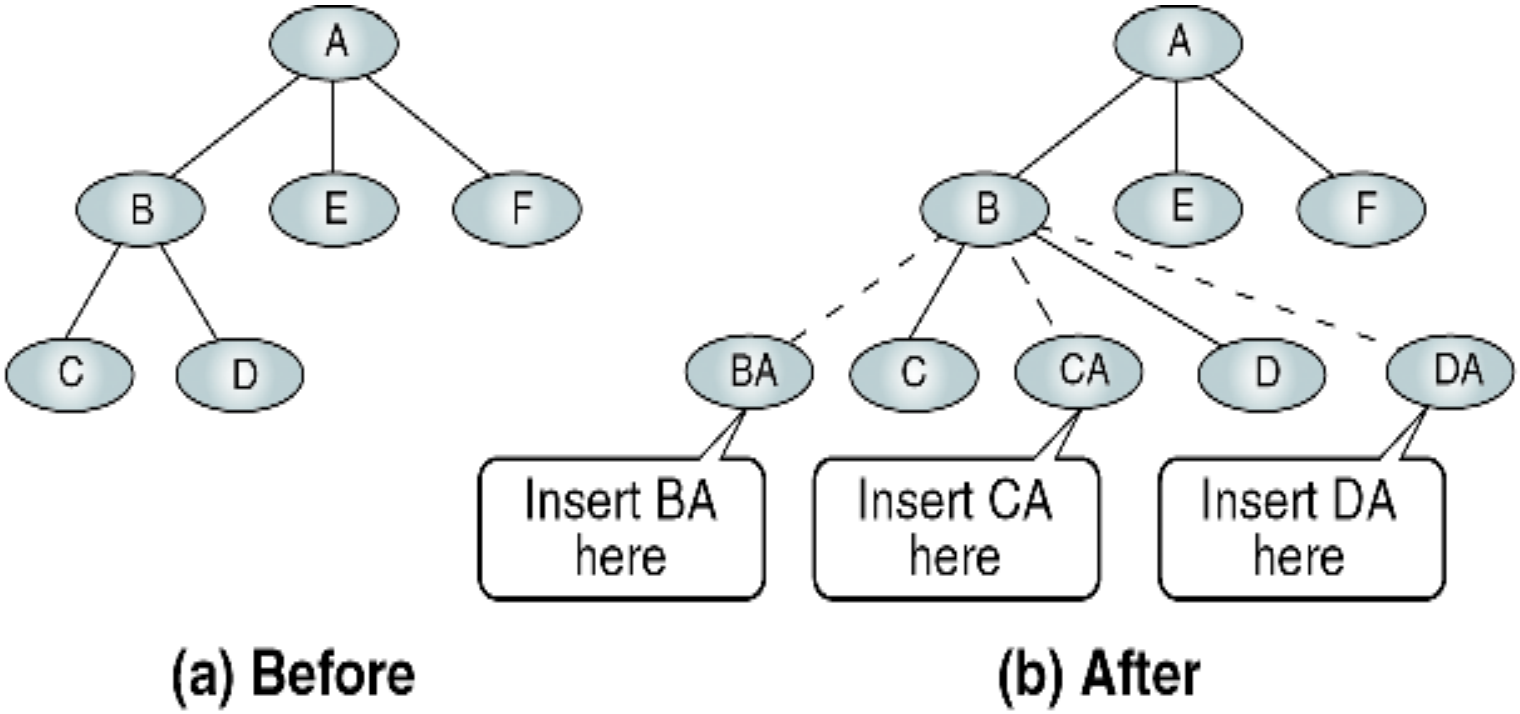
- To insert a node into a general tree, the user must supply the parent of the node



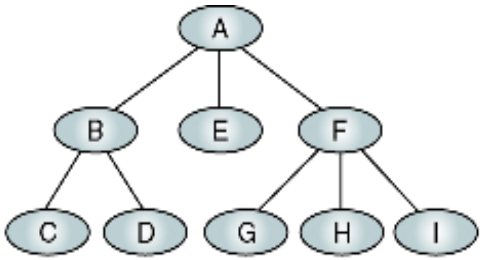
LIFO Insertion into General Trees



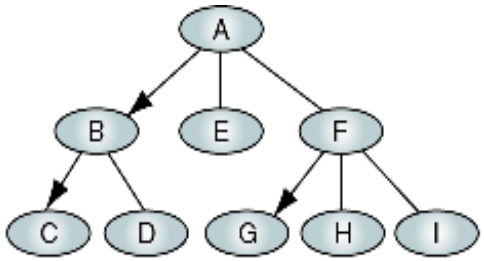
Key-sequenced insertion into General Tree



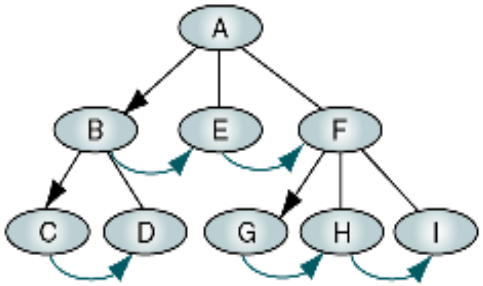
Converting General Trees to Binary Trees



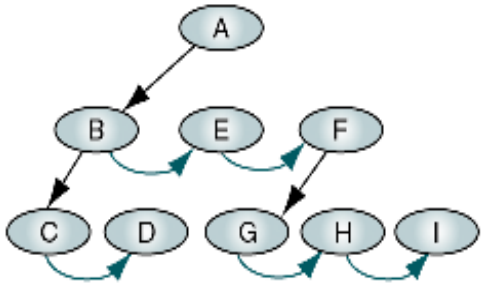
(a) The general tree



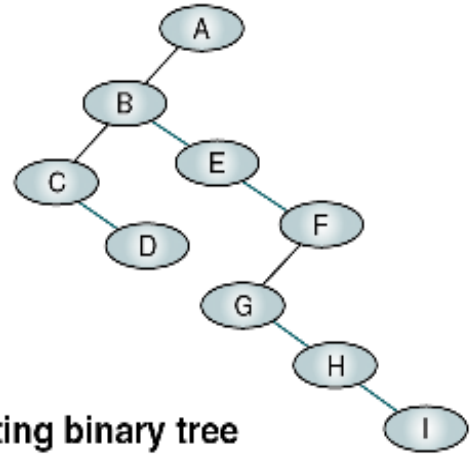
(b) Identify leftmost children



(c) Connect siblings



(d) Delete unneeded branches



(e) The resulting binary tree