

โครงสร้างข้อมูลแบบต้นไม้

อ.จรายา สายนุย

สอน. ค่าย 2 ปีการศึกษา 2560

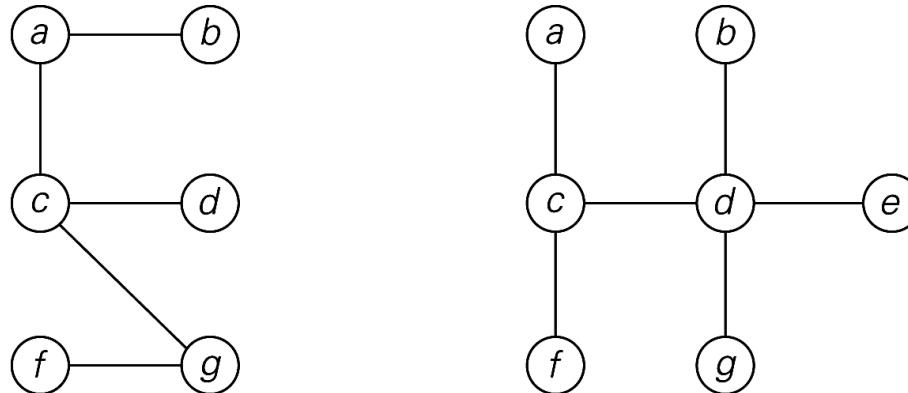
12 มีนาคม 2561

Learning Outcome

- มีความรู้ความเข้าใจนิยามและคุณสมบัติของโครงสร้างข้อมูลแบบต้นไม้
- เขียนโปรแกรมเพื่อเก็บข้อมูลโดยใช้โครงสร้างข้อมูลแบบต้นไม้แบบต่างๆ ได้
 - แบบอาร์เรย์
 - แบบพอยต์เตอร์
- เขียนโปรแกรมเพื่อดำเนินการบนโครงสร้างข้อมูลแบบต้นไม้ได้
 - การແວ່ພ່ານຕັ້ນໄມ້ແບບຕ່າງໆ
 - การນັບຈຳນວນໂທນດໃນຕັ້ນໄມ້
 - การหาຄວາມສູງຂອງຕັ້ນໄມ້

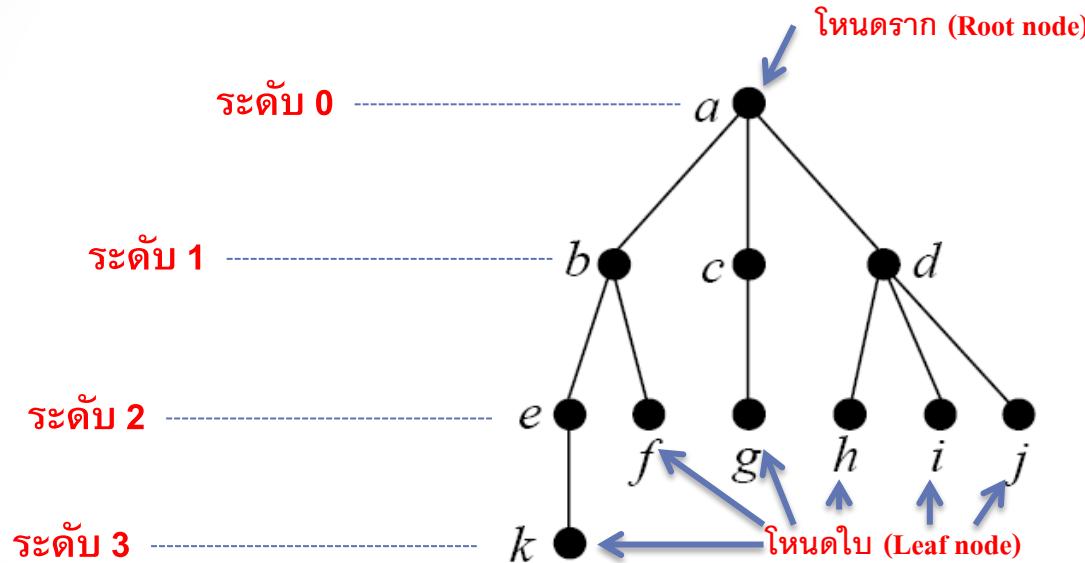
นิยามและสมบัติของต้นไม้

- ต้นไม้ (Tree) หมายถึง กราฟต่อเนื่องที่ไม่มีวัฏจักร
- คุณสมบัติของต้นไม้
 - มีเส้นทางเพียงเส้นเดียวจากโหนดหนึ่งไปยังอีกโหนดหนึ่ง
 - จำนวนเส้นเชื่อมในต้นไม้ เท่ากับ จำนวนโหนดของต้นไม้ลบด้วย 1



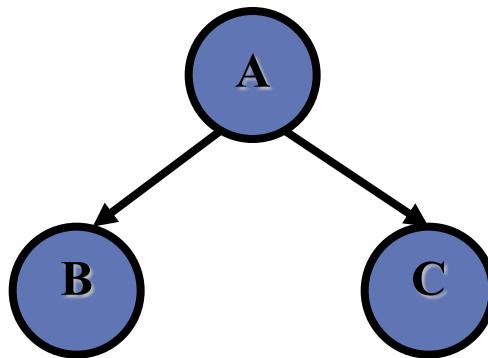
ตัวอย่างต้นไม้

ส่วนประกอบของต้นไม้แบบมีราก



Basic Tree Concepts

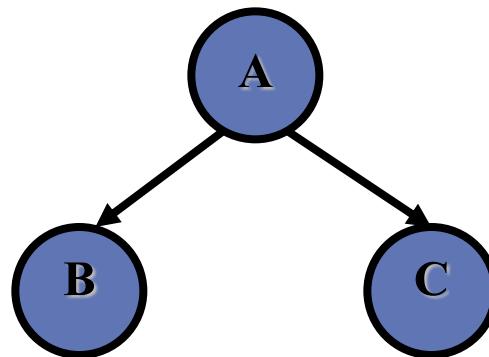
- Tree consists of
 - **nodes**: a finite set of elements
 - **branches**: a finite set of directed lines



Node	A, B, C
Branch	AB, AC

Basic Tree Concepts

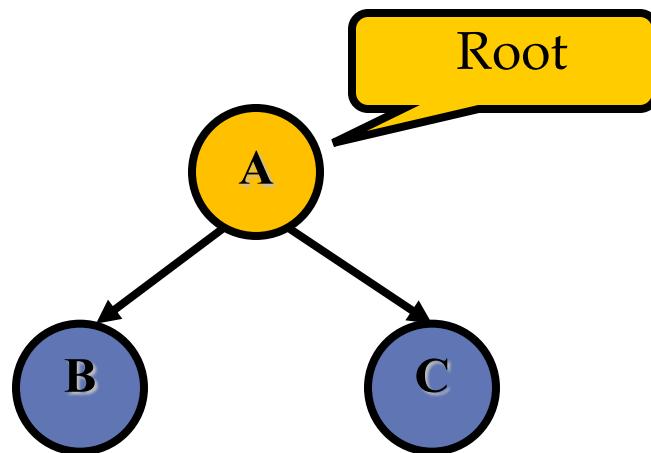
- **Degree of nodes** = number of branches associated with a node



Node A มี Degree เท่ากับ 2

Basic Tree Concepts

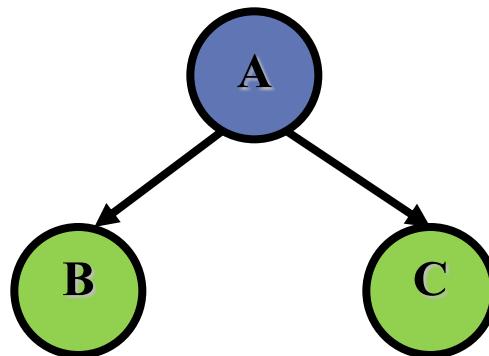
- **Root:** First node of a tree



Basic Tree Concepts (cont'd)

- Terminology

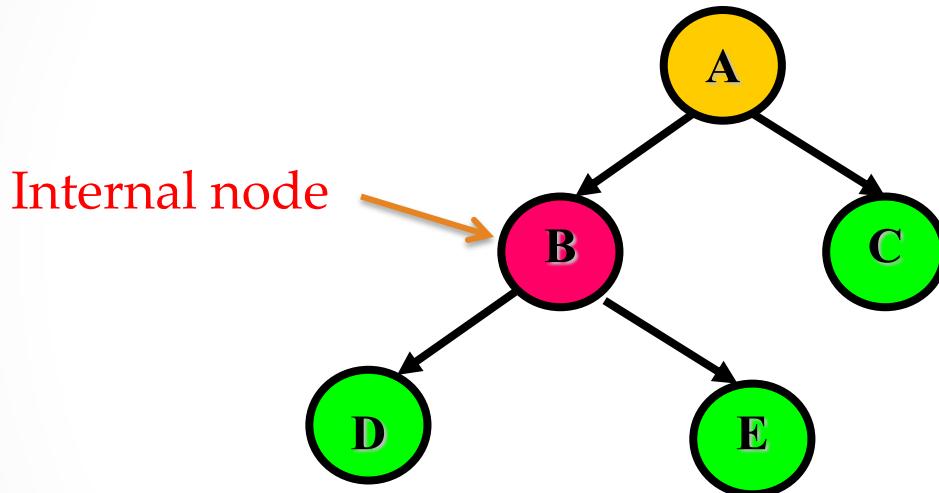
- **A leaf:** any node with an outdegree of zero or a node with no successor.



Leaf B, C

Basic Tree Concepts (cont'd)

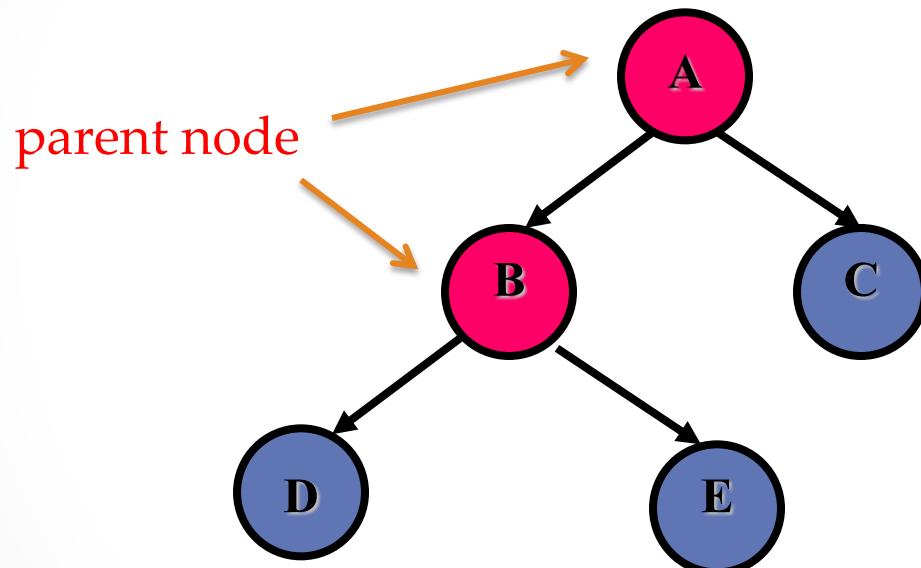
- An **internal node**: a node that is not a root or a leaf.



Root	A
Leaf	D, E, C
Internal node	B

Basic Tree Concepts (cont'd)

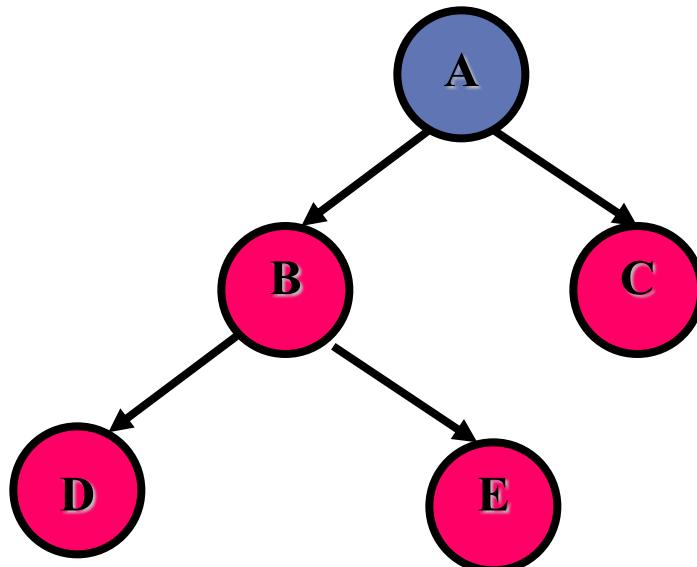
- A **parent**: a node that has successor nodes.



Parent
A, B

Basic Tree Concepts (cont'd)

- A **child**: a node with a predecessor.

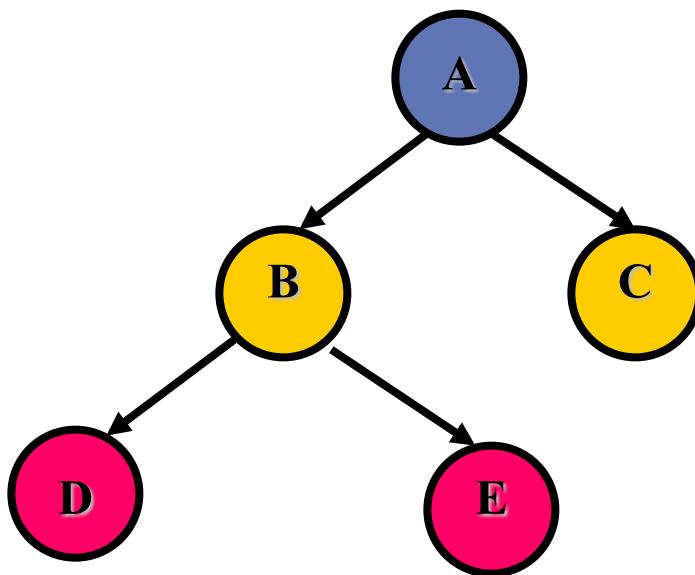


Child

B, C, D, E

Basic Tree Concepts (cont'd)

- **Siblings:** two or more nodes with the same parent

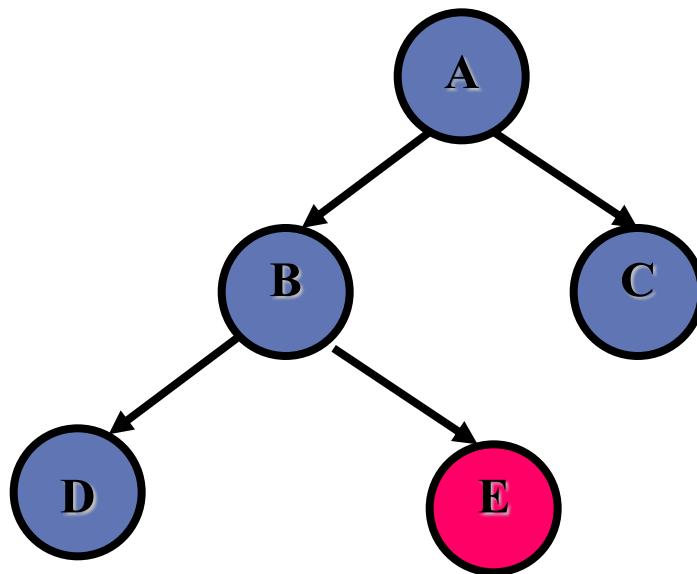


Sibling

{B, C}, {D, E}

Basic Tree Concepts (cont'd)

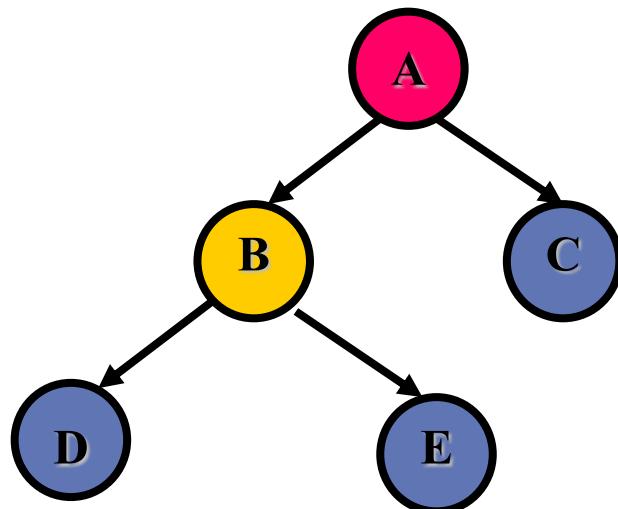
- An **ancestor**: any node in the path from the root to the node.



Ancestor $\forall \exists$ E A, B

Basic Tree Concepts (cont'd)

- A **descendant**: any node in the path below the parent node.

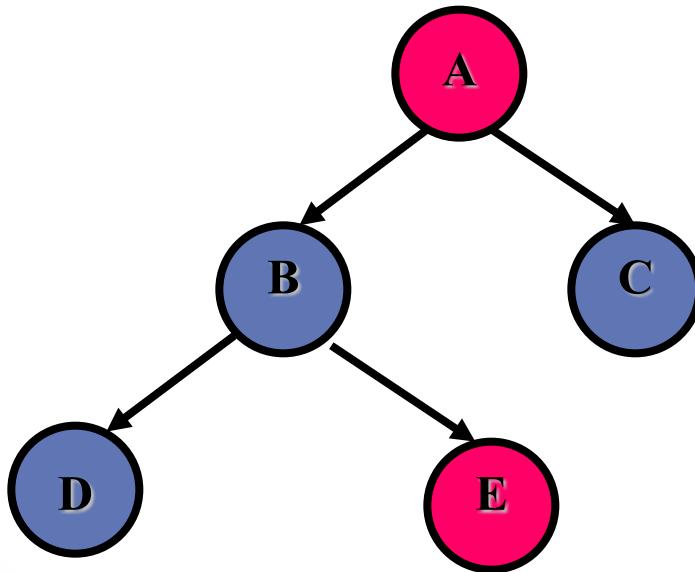


Descendent ឧទា A B, C, D, E

Descendent ឧទា B D,E

Basic Tree Concepts (cont'd)

- A **path**: a sequence of nodes in which each node is adjacent to the next one.

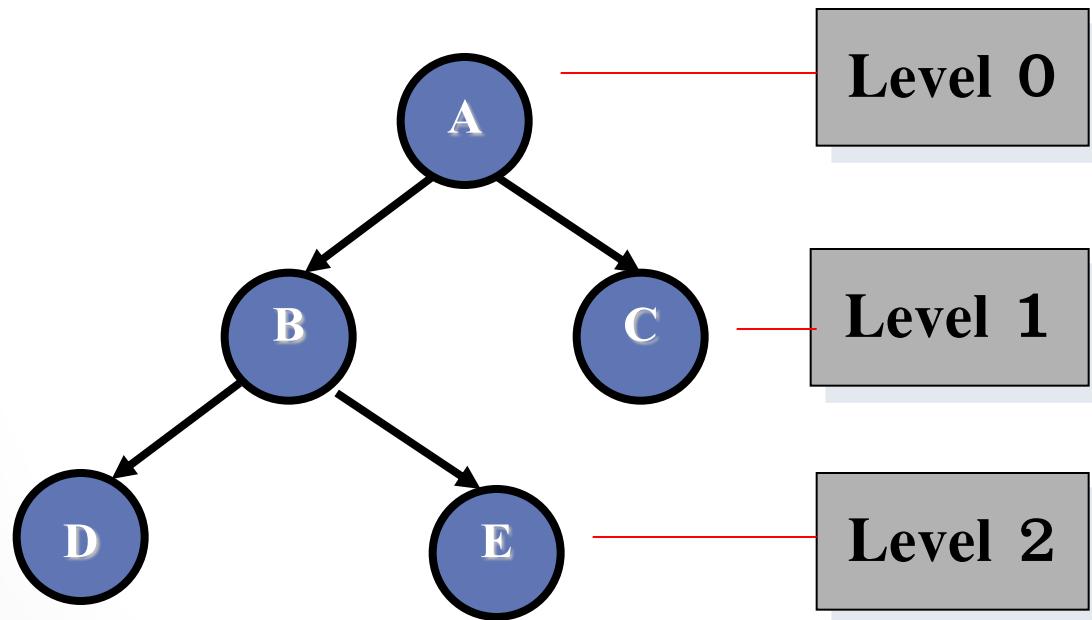


Path จาก A ไป E

A -> B -> E

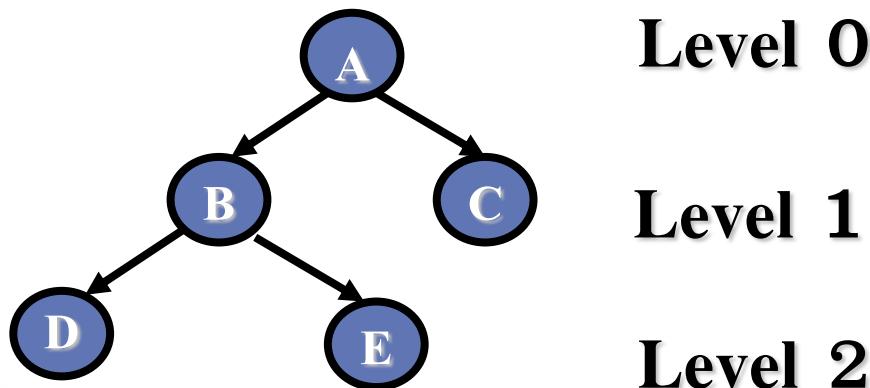
Basic Tree Concepts (cont'd)

- The **level** of a node: the distance from the root.



Basic Tree Concepts (cont'd)

- The **height** of the tree: the level of the leaf in the longest path from the root.
- The height of an empty tree = -1 (By definition)



Height = 2

Binary Tree

...

ต้นไม้ทวิภาค

ต้นไม้ทวิภาค

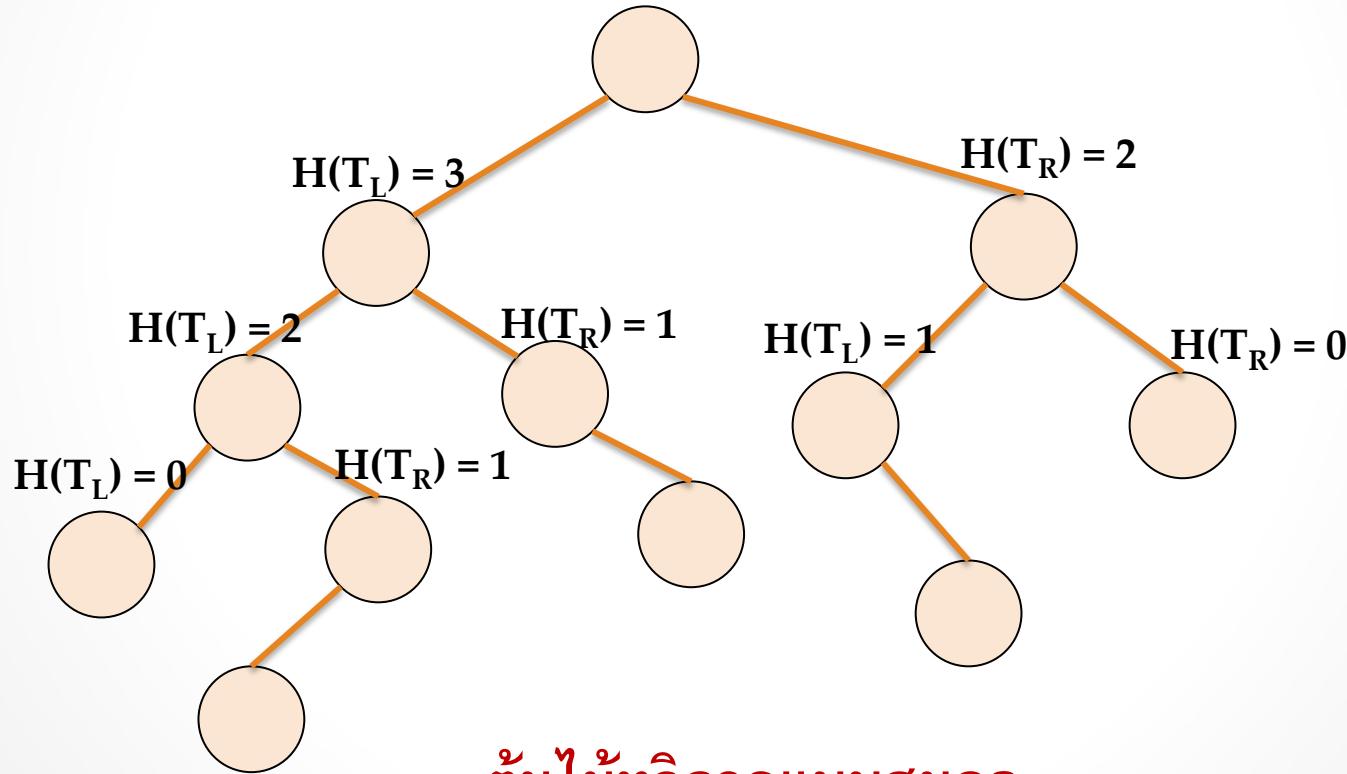
- ต้นไม้ทวิภาค (Binary Tree) — ต้นไม้แบบมีราก ที่หนนดใดๆ มีโหนดลูกไม่เกิน 2 โหนด
- ประเภทของต้นไม้ทวิภาค

ต้นไม้ทวิภาคแบบเต็ม	ต้นไม้ทวิภาคแบบสมบูรณ์	ต้นไม้ทวิภาคแบบสมดุล
<pre>graph TD; 25((25)) --- 67((67)); 25 --- 48((48)); 67 --- 54((54)); 67 --- 12((12)); 12 --- 43((43)); 12 --- 72((72)); 48 --- 38((38)); 48 --- 21((21));</pre> <p>โหนดทุกโหนดยกเว้นโหนดใบ มีดีกรี <u>เท่ากับ 2</u></p>	<pre>graph TD; 25((25)) --- 67((67)); 25 --- 48((48)); 67 --- 54((54)); 67 --- 12((12)); 54 --- 66((66)); 54 --- 95((95)); 12 --- 2((2)); 12 --- 16((16)); 48 --- 38((38)); 48 --- 21((21)); 38 --- 62((62)); 38 --- 35((35)); 21 --- 59((59)); 21 --- 73((73));</pre> <p>โหนด <u>ใบ</u>ทุกโหนดอยู่ในระดับเดียวกัน</p>	<pre>graph TD; 25((25)) --- 67((67)); 25 --- 48((48)); 67 --- 54((54)); 67 --- 12((12)); 48 --- 43((43)); 48 --- 21((21)); 54 --- 66((66)); 54 --- 95((95)); 12 --- 5((5)); 12 --- 17((17)); 95 --- 33((33));</pre> <p>ความสูงของต้นไม้ย่อทางซ้าย และทางขวาของโหนดใดๆ ต่างกัน <u>ไม่เกิน 1</u></p>

การตรวจสอบสมดุลต้นไม้

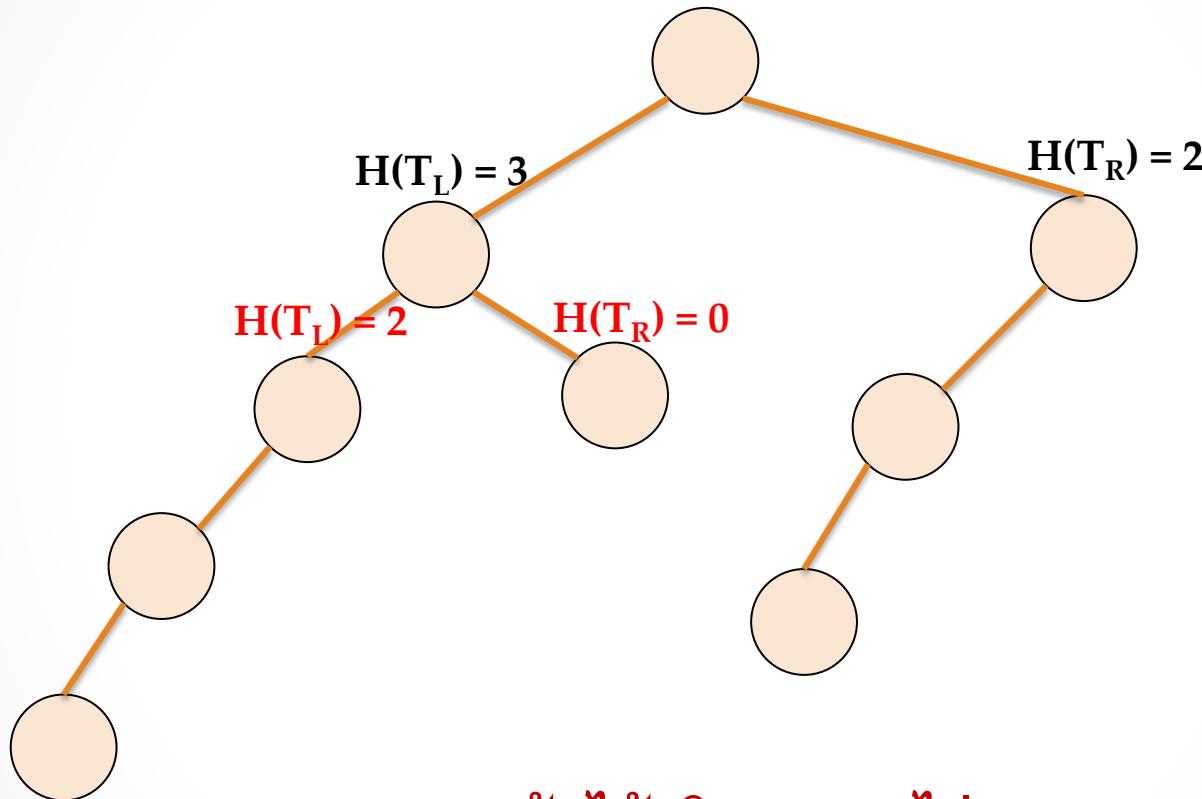
ความสูงของต้นไม้ย่อทางซ้ายและทางขวาของโหนดใดๆ ต่างกัน ไม่เกิน 1

$$|H(T_L) - H(T_R)| \leq 1$$



ต้นไม้ที่วิภาคแบบสมดุล

การตรวจสอบสมดุลต้นไม้



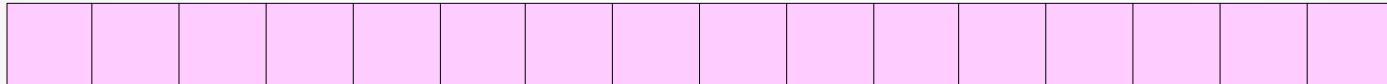
ต้นไม้ทวิภาคแบบไม่สมดุล

โครงสร้างของต้นไม้ทวิภาค

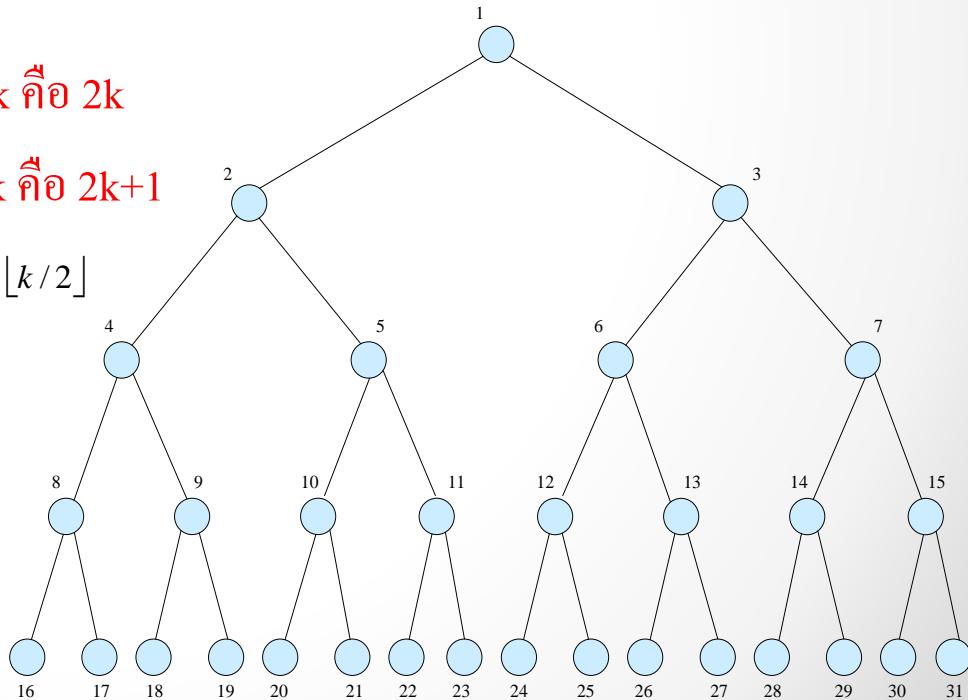
- แบบหน่วยเก็บต่อเนื่อง (continuous storage)
- แบบแคลวลำดับ (array-based)
- แบบตัวชี้ (pointer-based)

ແບບໜ່ວຍເກີບຕ້ອນເນື້ອງ

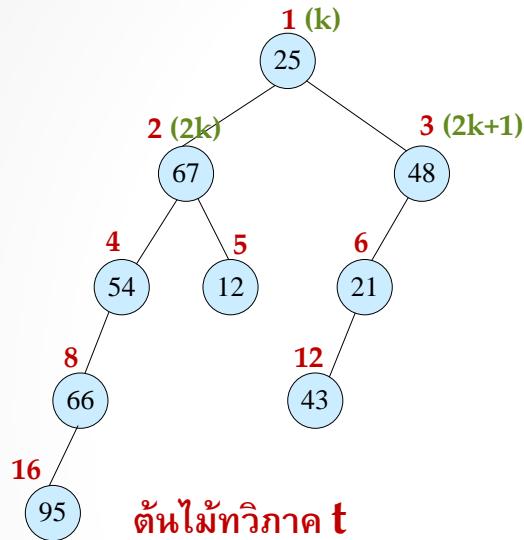
0 1 2 3 4 5 6 7 8 9 ... 12 ... 16 ... 31



- ใช้ແກວລຳດັບ 1 ມິຕີ ສໍາຫຼັບເກີບຂໍ້ມູນ
- ໂດຍຕຳແໜ່ງຂອງແຕ່ລະໂທນດໃນແກວລຳດັບເປັນດັ່ງນີ້
 - ຕຳແໜ່ງທີ 1 ເປັນໂທນດຮາກ
 - ຕຳແໜ່ງໂທນດລູກທາງຊ້າຍຂອງໂທນດ k ຄື່ອ $2k$
 - ຕຳແໜ່ງໂທນດລູກທາງຂວາຂອງໂທນດ k ຄື່ອ $2k+1$
 - ຕຳແໜ່ງໂທນດພ່ອແມ່ນຂອງໂທນດ k ຄື່ອ $\lfloor k/2 \rfloor$



แบบหน่วยเก็บต่อเนื่อง (cont'd)



ตัวอย่าง การประกาศโครงสร้างข้อมูล

```
#define MAXNODES 32  
int t[MAXNODES];
```



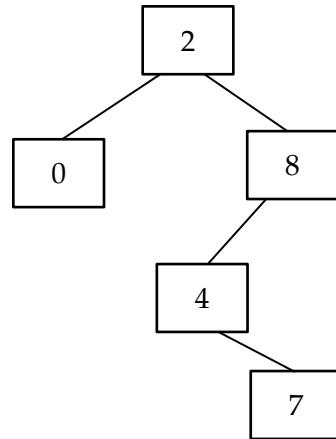
สิ่นเปลืองเนื้อที่ในกรณีที่หนดส่วนใหญ่เป็นหนดว่าง

0	1	2	3	4	5	6	7	8	9	...	12	...	16	...	31
9	25	67	48	54	12	21		66		...	43	...	95	...	

โครงสร้างแบบหน่วยเก็บต่อเนื่องของต้นไม้ทวิภาค t

แบบฝึกหัด

- แปลงจาก binary tree เป็นโครงสร้างหน่วยเก็บแบบต่อเนื่อง



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	...	[13]

โครงสร้างแบบหน่วยเก็บต่อเนื่องของต้นไม้ทวิภาค t

แบบแطرอล์ดับ

- ใช้แطرอล์ดับ 1 มิติ สำหรับเก็บข้อมูล
- ไม่มีการกำหนดตำแหน่งที่แน่นอน
- แต่ละโหนดเก็บข้อมูล 3 ค่า ได้แก่
 1. ตำแหน่งโหนดลูกทางซ้าย
 2. ตำแหน่งโหนดลูกทางขวา
 3. ข้อมูล



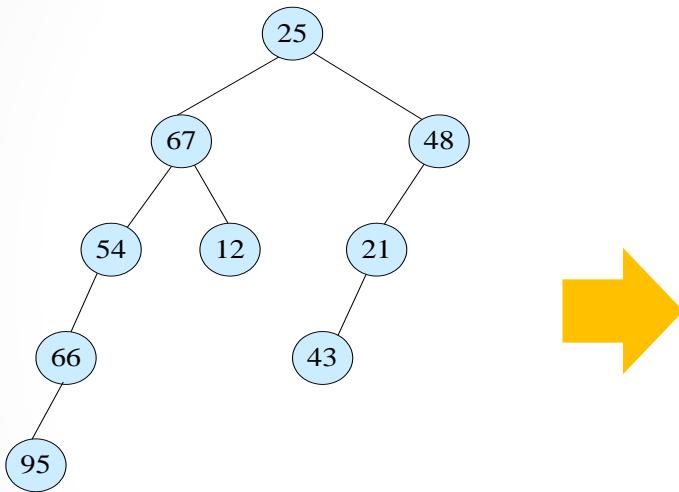
แบบແກວລຳດັບ (cont'd)

- ตัวอย่างการประการໂຄງສ້າງຂໍ້ມູນແບບແກວລຳດັບ

```
#define MAXNODES 32
typedef struct {
    int left; //ເກີບຕຳແໜ່ງໂທນດລູກທາງຊ້າຍ
    int data;
    int right; //ເກີບຕຳແໜ່ງໂທນດລູກທາງຂວາ
} node;
typedef struct {
    int rootIndex; //ເກີບຕຳແໜ່ງໂທນດຮາກ
    int freeListIndex;
    node table[MAXNODES];
} binaryTree;
binaryTree t;
```

binaryTree t			
	rootIndex	-1	
	freeListIndex	0	
[]	left	data	right
0	-1	0	1
1	-1	0	2
2	-1	0	3
3	-1	0	4
4	-1	0	5
5	-1	0	6
6	-1	0	7
7	-1	0	8
8	-1	0	9
9	-1	0	10
.	.	.	.
.	.	.	.
29	-1	0	30
30	-1	0	-1

แบบແລວລຳດັບ (cont'd)

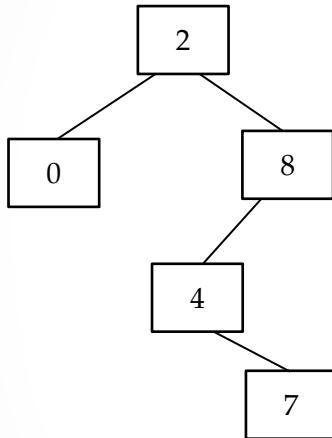


ต้นໄນ້ກວດຄາຕ
ຫັ້ງຈາກເພີມໂທນດ
25 67 48 21 54 12 43 66 95 ຕາມລຳດັບ

binaryTree t			
rootIndex	0	freeListIndex	9
table			
[]	left	data	right
0	1	25	2
1	4	67	5
2	3	48	-1
3	6	21	-1
4	7	54	-1
5	-1	12	-1
6	-1	43	-1
7	8	66	-1
8	-1	95	-1
9	-1	0	10
.	.	.	.
.	.	.	.
29	-1	0	30
30	-1	0	-1

แบบฝึกหัด

- แปลงจาก binary tree เป็นโครงสร้างหน่วยเก็บแบบแคล้มดับ



เพิ่มโหนด 2, 0, 8, 4, 7 ตามลำดับ

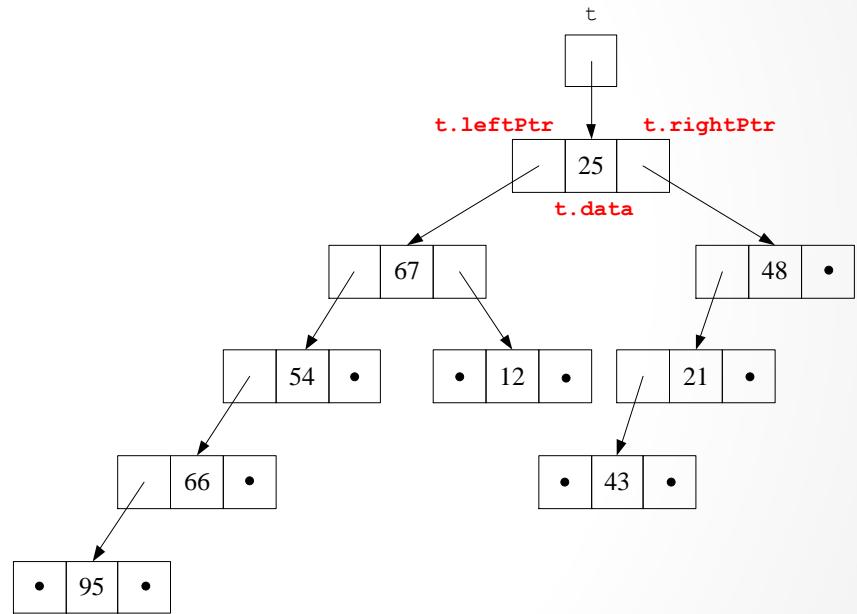
currIdx	0
---------	---

t[]	left	data	right
0			
1			
2			
3			
5			

แบบตัวชี้

- ตัวอย่างการประกาศโครงสร้างข้อมูลแบบตัวชี้

```
struct node {  
    struct node *leftPtr;  
    int      data;  
    struct node *rightPtr;  
};  
  
typedef struct node BTNode;  
typedef BTNode *binaryTree;  
  
binaryTree t;
```



โครงสร้างแบบตัวชี้ของต้นไม้ทวิภาค t

แบบฝึกหัด

1. เขียนโปรแกรมเพื่อเก็บข้อมูลไว้ในโครงสร้างแบบต้นไม้ทวิภาค โดยใช้โครงร่าง 3 แบบ

- 1.1 แบบหน่วยเก็บต่อเนื่อง พร้อมพิมพ์ค่าที่อยู่อาศัย
- 1.2 แบบแคลล์ดับ พร้อมพิมพ์ค่าที่อยู่ในตัวแปรต่างๆ
- 1.3 แบบตัวชี้

	rootIndex:	0	
	freeListIndex:	9	
[]	left	data	right
0	1	25	6
1	2	67	5
2	3	54	-1
3	4	66	-1
4	-1	95	-1
5	-1	12	-1
6	7	48	-1
7	8	21	-1
8	-1	43	-1
9	-1	0	10
10	-1	0	11
11	-1	0	12
12	-1	0	13
13	-1	0	14
14	-1	0	15
15	-1	0	16
16	-1	0	17
17	-1	0	18
18	-1	0	19
19	-1	0	20
20	-1	0	21
21	-1	0	22
22	-1	0	23
23	-1	0	24
24	-1	0	25
25	-1	0	26
26	-1	0	27
27	-1	0	28
28	-1	0	29
29	-1	0	30
30	-1	0	31
31	-1	0	-1

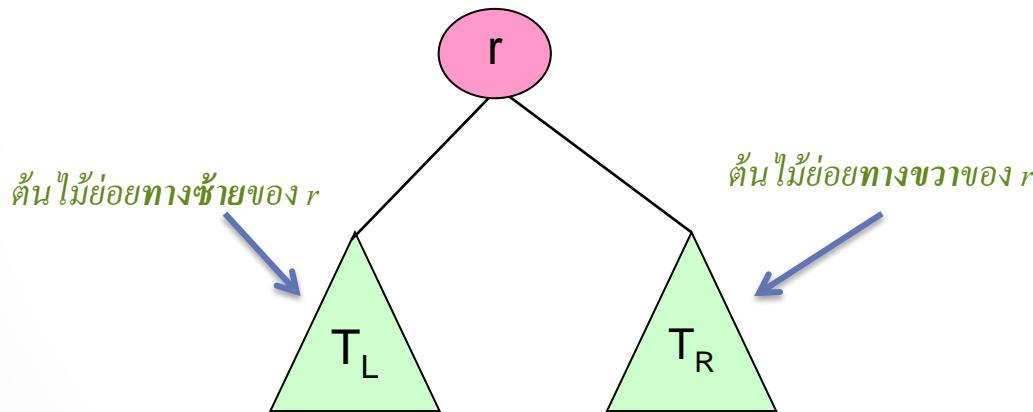
การดำเนินการกับต้นไม้ทั่วภาค

- การนับจำนวนโหนด
- การหาความสูง
- การตรวจสอบต้นไม้

การนับจำนวนโหนด

- นิยามจำนวนโหนดของต้นไม้ $T(r)$ เขียนได้ดังนี้

$$N(T(r)) = \begin{cases} 0 & \text{if } r = \text{nil} \\ 1 + N(T(L(r))) + N(T(R(r))) & \text{if } r \neq \text{nil} \end{cases}$$



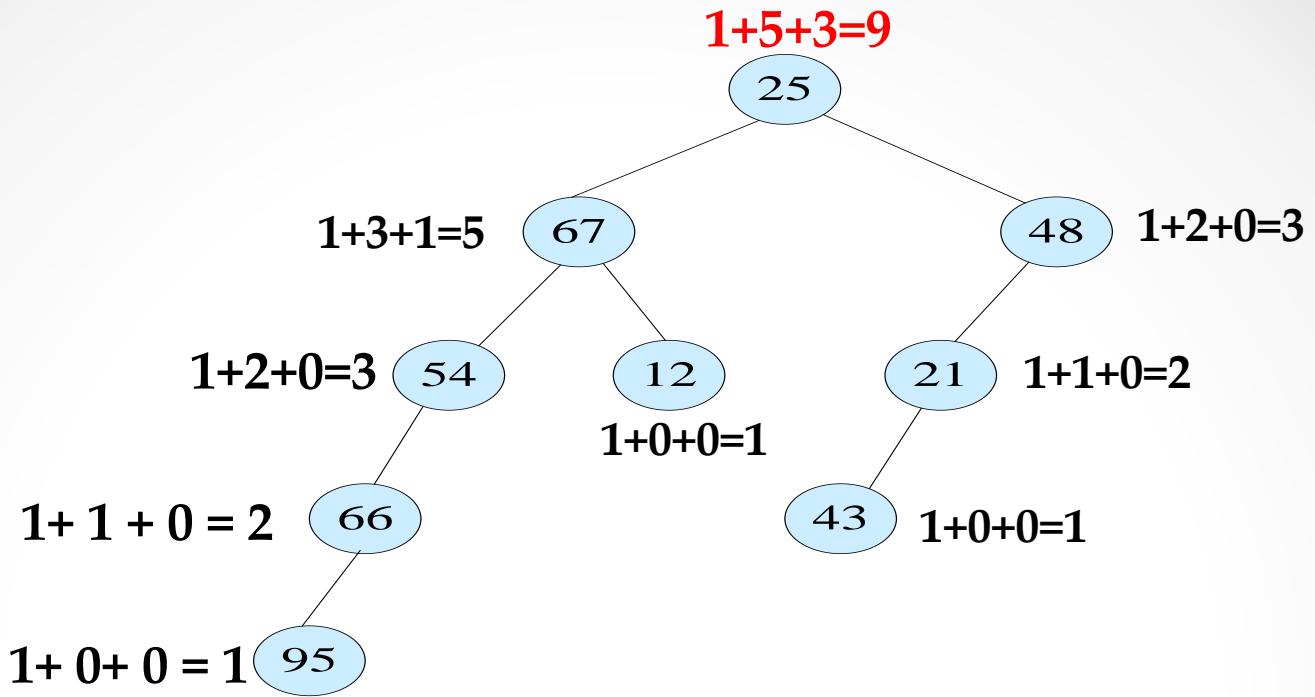
$T(r)$ ต้นไม้ที่มีโหนด r เป็นราก

การนับจำนวนโหนด

- จากนิยาม เขียนฟังก์ชันการนับจำนวนโหนด ได้ดังนี้

$$N(T(r)) = \begin{cases} 0 & \text{if } r = \text{nil} \\ 1 + N(T(L(r))) + N(T(R(r))) & \text{if } r \neq \text{nil} \end{cases}$$

```
FUNCTION Size( R : BinaryTree ) : integer;  
BEGIN  
    IF R = nil THEN  
        Size := 0  
    ELSE  
        Size := 1 + Size(R.Left) + Size(R.Right));  
    END;
```



$$\text{Size}(T(25)) = 1 + \text{Size}(T(67)) + \text{Size}(T(48)) = 9$$

$$\text{Size}(T(67)) = 1 + \text{Size}(T(54)) + \text{Size}(T(12)) = 5$$

$$\text{Size}(T(54)) = 1 + \text{Size}(T(66)) + 0 = 3$$

$$\text{Size}(T(66)) = 1 + \text{Size}(T(95)) + 0 = 2$$

$$\text{Size}(T(95)) = 1 + 0 + 0 = 1$$

$$\text{Size}(T(12)) = 1 + 0 + 0 = 1$$

$$\text{Size}(T(48)) = 1 + \text{Size}(T(21)) + 0 = 3$$

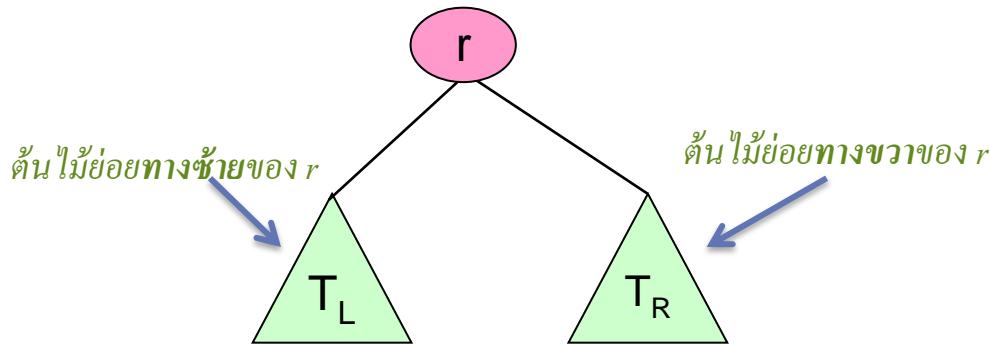
$$\text{Size}(T(21)) = 1 + \text{Size}(T(43)) + 0 = 2$$

$$\text{Size}(T(43)) = 1 + 0 + 0 = 1$$

การหาความสูง

นิยามการหาความสูงของต้นไม้ $T(r)$ เขียนได้ดังนี้

$$H(T(r)) = \begin{cases} -1 & \text{if } r = nil \\ 1 + \max(H(T(L(r))), H(T(R(r)))) & \text{if } r \neq nil \end{cases}$$



$T(r)$ ต้นไม้ที่มีโหนด r เป็นราก

การหาความสูง

- จากนิยาม เขียนฟังก์ชันการนับจำนวนโหนด ได้ดังนี้

$$H(T(r)) = \begin{cases} -1 & \text{if } r = \text{nil} \\ 1 + \max(H(T(L(r))), H(T(R(r)))) & \text{if } r \neq \text{nil} \end{cases}$$

```
FUNCTION Height( R : BinaryTree ) : integer;
BEGIN
  IF R = nil THEN
    Height := -1
  ELSE
    Height := 1 + Max(Height(R.Left), Height(R.Right));
END;
```

ลำดับการเรียกฟังก์ชัน $H(r)$

$$H(25) = 1 + \max(H(67), H(48)) = 1 + 3 = 4$$

$$H(67) = 1 + \max(H(54), H(12)) = 1 + 2 = 3$$

$$H(54) = 1 + \max(H(66), -1) = 1 + 1 = 2$$

$$H(66) = 1 + \max(H(95), -1) = 1 + 0 = 1$$

$$H(95) = 1 + \max(-1, -1) = 0$$

$$H(12) = 1 + \max(-1, -1) = 0$$

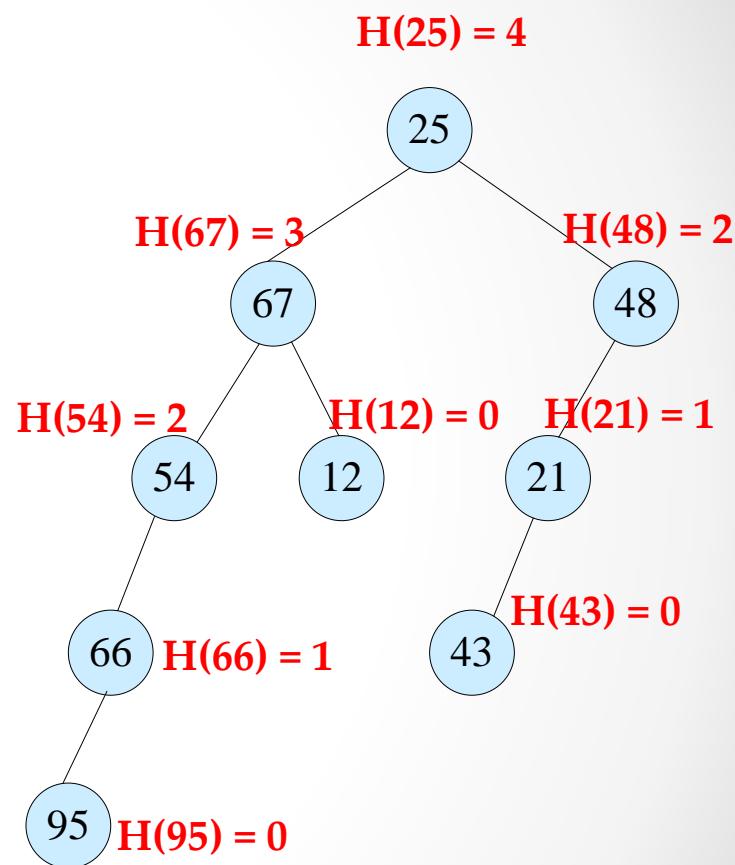
$$H(48) = 1 + \max(H(21), -1) = 2$$

$$H(21) = 1 + \max(H(43), -1) = 1$$

$$H(43) = 1 + \max(-1, -1) = 1 - 1 = 0$$

* ความสูงของโนนด์ใบเท่ากับ 0

** ความสูงของต้นไม้จั่งเท่ากับ -1



การดำเนินการกับต้นไม้ทวิภาค

- การແວແຜ່ງຕົ້ນໄມ້

- **Preorder traversal (NLR):**

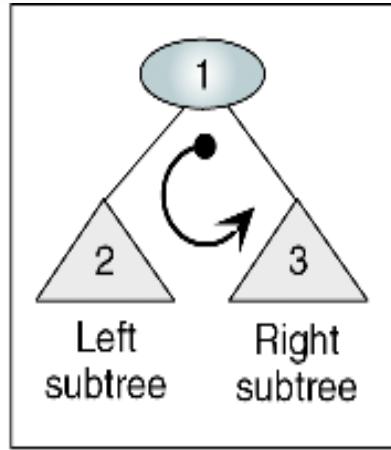
- process root (N) -> left subtree (L) -> right subtree (R)

- **Inorder traversal (LNR):**

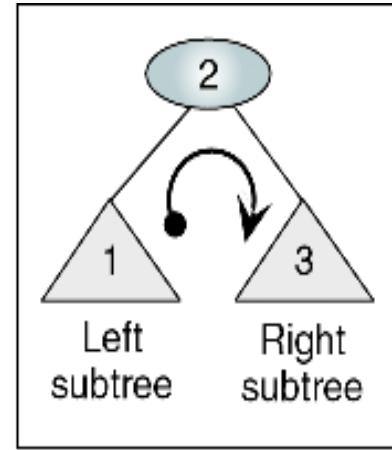
- process left subtree (L) -> the root (N) -> right subtree (R)

- **Postorder traversal (LRN):**

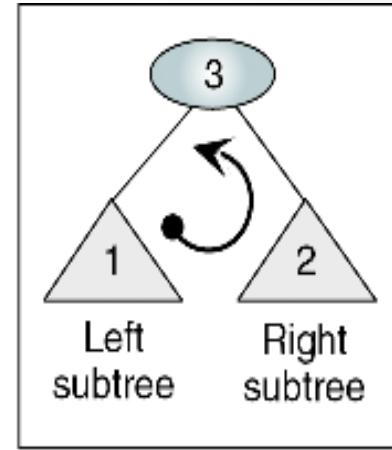
- process left subtree (L) -> right subtree (R) -> root (N)



(a) Preorder traversal



(b) Inorder traversal

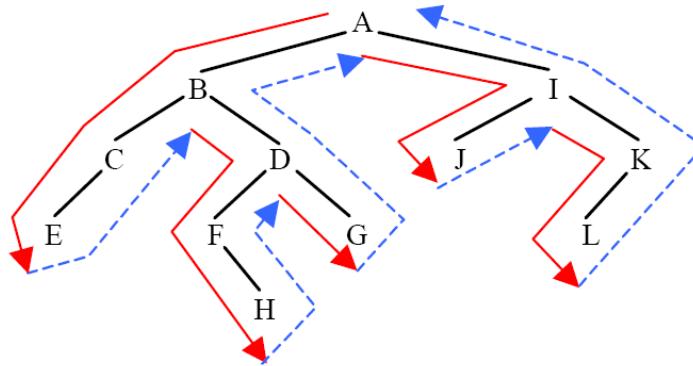


(c) Postorder traversal

FIGURE 6-8 Binary Tree Traversals

การແວແນບກອນລຳດັບ

$$Pre(T(r)) = \begin{cases} \text{empty sequence} & \text{if } r = \text{nil} \\ r, \ Pre(T(L(r))), \ Pre(T(R(r))) & \text{if } r \neq \text{nil} \end{cases}$$



ລຳດັບການແວແນບ **preorder**: **A B C E D F H G I J K L**

(**NLR**):

การແວະຜ່ານແບນກ່ອນລຳດັບ

ຈາກນິຍາມ ເຂີຍເປັນພັງກົນ ໄດ້ດັ່ງນີ້

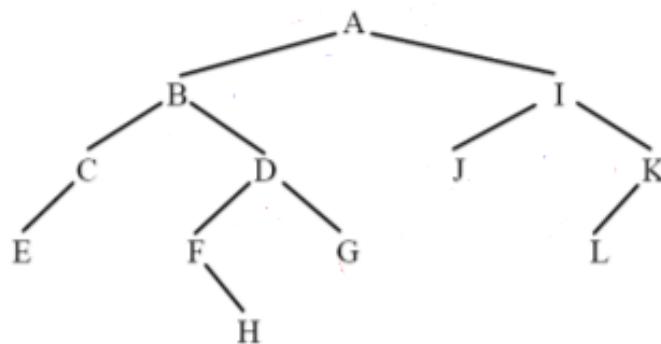
```
PROCEDURE PreOrder( R : BinaryTree )
BEGIN
    IF R <> nil THEN
        BEGIN
            Visit( r );
            PreOrder( R^.Left );
            PreOrder( R^.Right );
        END;
    END;
```

ເພື່ອດຳເນີນການ ເຊັ່ນ

- ພິມພົດ
- ເປົ້າຍເຖິງ

การແວ່ຜ່ານແບບຕາມລຳດັບ

$$In(T(r)) = \begin{cases} \text{empty sequence} & \text{if } r = \text{nil} \\ In(T(L(r))), r, In(T(R(r))) & \text{if } r \neq \text{nil} \end{cases}$$



ລຳດັບການແວ່ແບບ **inorder**: **E C B F H D G A J I L K**

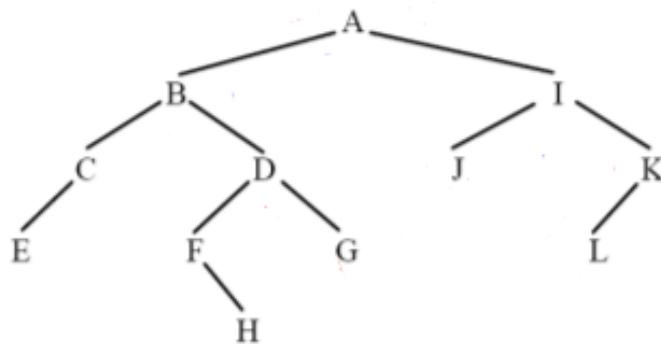
(LNR):

การແວ່ພ່ານແບນຕາມລຳດັບ

```
PROCEDURE InOrder( R : BinaryTree )
BEGIN
    IF R <> nil THEN
        BEGIN
            InOrder( R^.Left );
            Visit( R );
            InOrder( R^.Right );
        END;
    END;
```

การແຈ່ງຜ່ານແບນທັງລຳດັບ

$$Post(T(r)) = \begin{cases} \text{empty sequence} & \text{if } r = \text{nil} \\ Post(T(L(r))), Post(T(R(r))), r & \text{if } r \neq \text{nil} \end{cases}$$



ລຳດັບການແຈ່ງຜ່ານແບນ **postorder**: **E C H F G D B J L K I A**

(LRN)

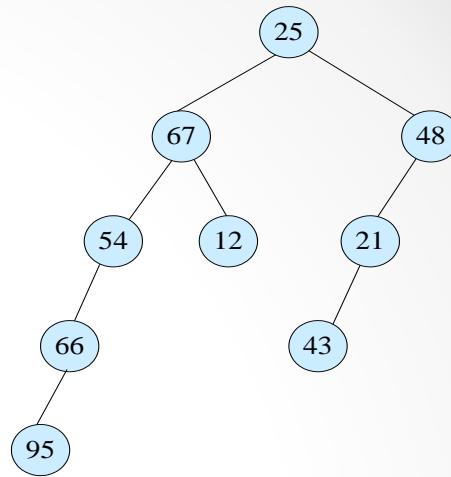
การเว้นผ่านแบบหลังลำดับ

```
PROCEDURE PostOrder( R : BinaryTree )
BEGIN
    IF R <> nil THEN
        BEGIN
            PostOrder( R^.Left );
            PostOrder( R^.Right );
            Visit( R );
        END;
    END;
```

แบบฝึกหัด

2. จากโปรแกรมในข้อ 1.3 เพิ่มส่วนของโปรแกรมเพื่อ

- นับจำนวนโหนด
- หาความสูงของต้นไม้
- พิมพ์โหนดทั้งหมดที่ແວ່ມຳຕາມລຳດັບ โดยໃຊ້ກາຣແວ່ມຳ 3 ແບບ
 - ແບບກ່ອນລຳດັບ (**preorder** traversal)
 - ແບບຕາມລຳດັບ (**inorder** traversal)
 - ທັງລຳດັບ (**postorder** traversal)



```
----- Display -----  
Number of Nodes: 9  
Height of Tree: 4  
PreOrder: 25 67 54 66 95 12 48 21 43  
InOrder: 95 66 54 67 12 25 43 21 48  
PostOrder: 95 66 54 12 67 43 21 48 25
```

Binary Tree Application

1. Expression Tree
2. Huffman Code

Expression Tree

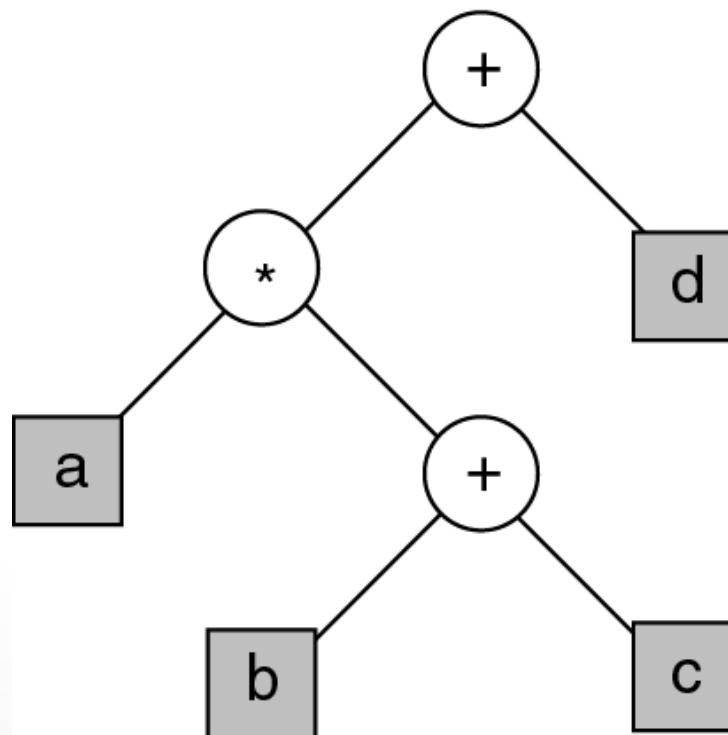
Binary Tree ที่มีคุณสมบัติดังต่อไปนี้

1. Leaf เก็บ Operand
2. Root และ Internal node เก็บ Operator
3. Subtree เป็น Sub expression

Expression Tree

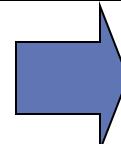
Example

a * (b + c) + d



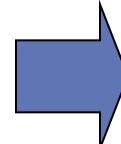
Expression Tree Traversal

1. Preorder Traversal



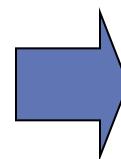
Prefix Expression

2. Postorder Traversal



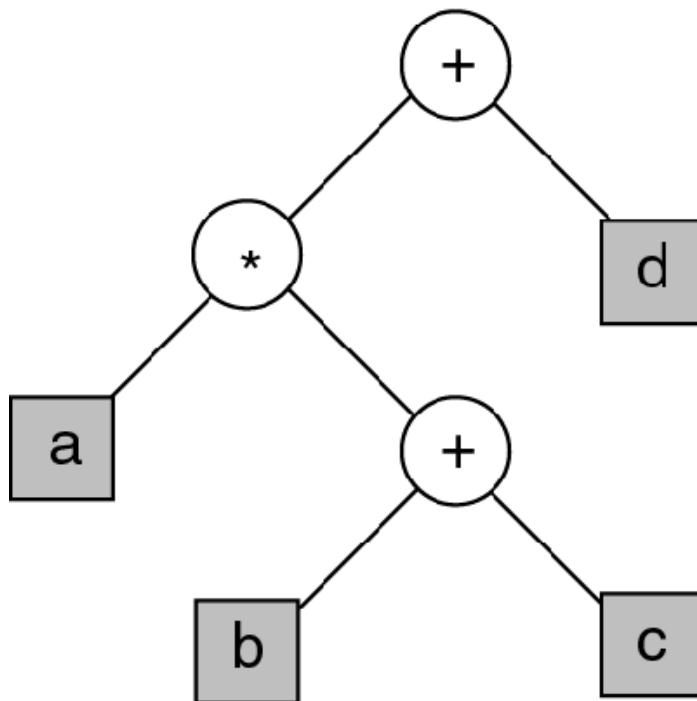
Postfix Expression

3. Inorder Traversal



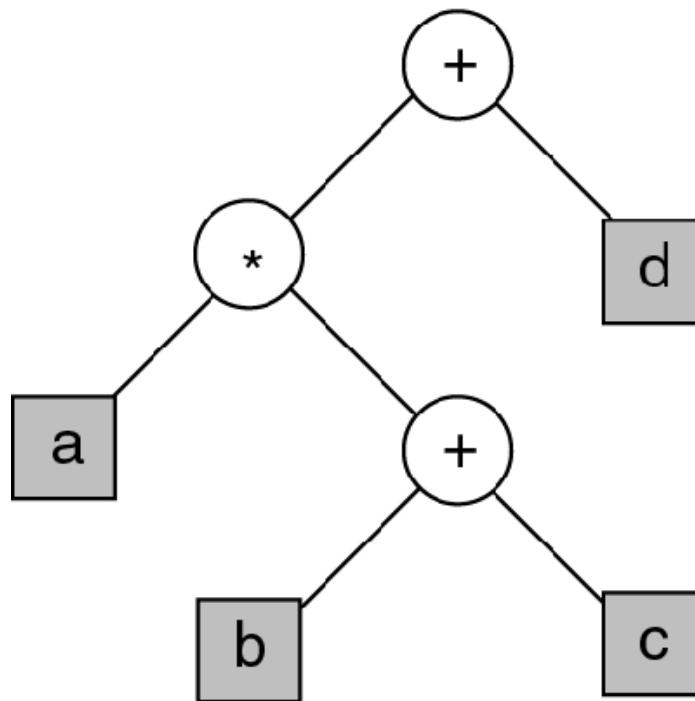
Infix Expression

Preorder Traversal



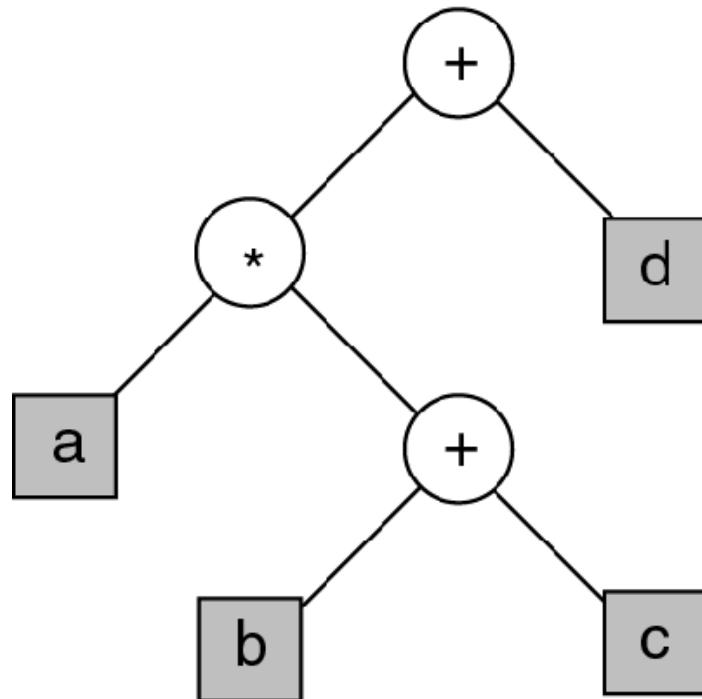
+ * a + b c d

Postorder Traversal



a b c + * d +

Inorder Traversal



a * b + c + d

Huffman code

- บีบอัด file โดยไม่ให้เสียข้อมูลไปเลยได้อย่างไร
- ให้ M เป็น ข้อมูลที่เราต้องการบีบอัด สมมติว่ามันมีขนาด 100,000 characters ซึ่งประกอบไปด้วยตัวอักษร a ถึง e เท่านั้น
- สมมติให้ 1 character ใช้ 1 byte (8 bits)
- จะนั้น 100,000 characters ใช้ 800,000 bits
- เราสามารถลดจำนวน bits ที่ใช้แต่ละตัวอักษรได้หรือไม่?

Huffman code

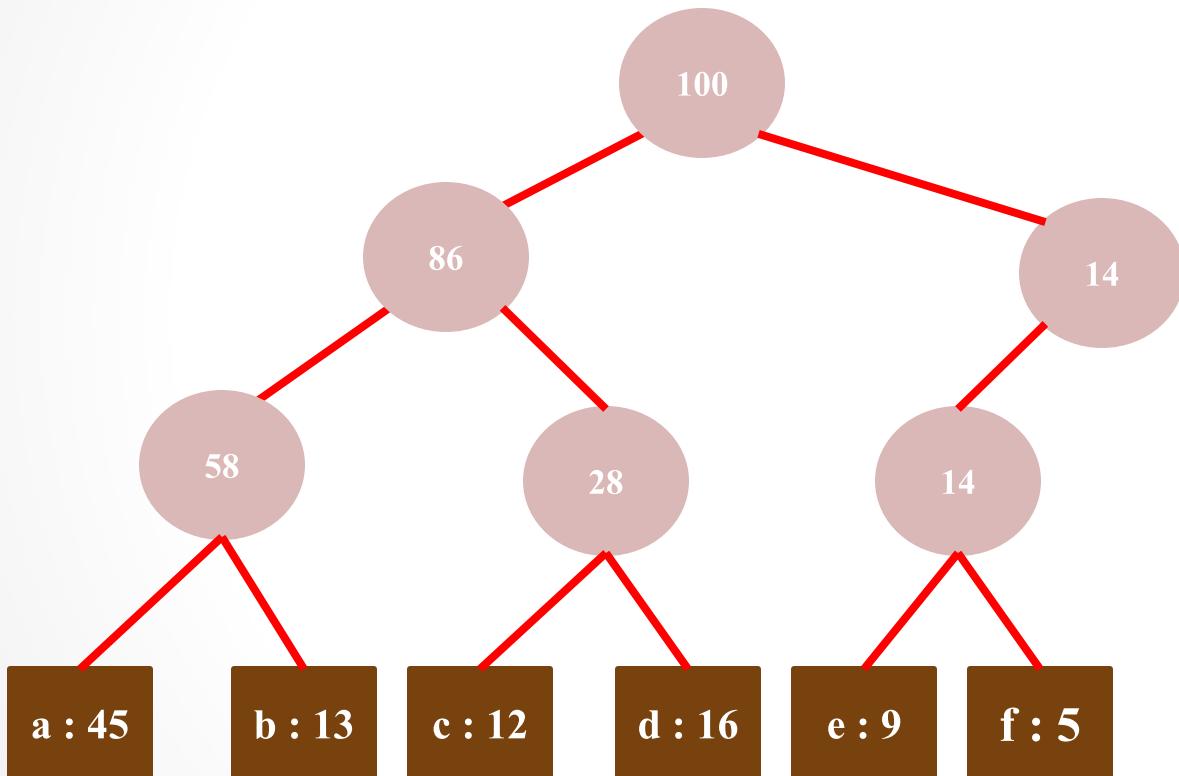
- เป็นขั้นตอนวิธีการลดขนาดของข้อมูลที่มีจำนวนมากโดยที่สามารถนำข้อมูลแปลงกลับมาในสภาพที่เหมือนเดิมได้
- หลักการคือการใช้ตารางความถี่การปรากฏของแต่ละตัวอักษร
- เป็นวิธีการแทนที่ตัวหนังสือ ที่น้อยที่สุดโดยใช้ Binary String
- โดยทั่วไปสามารถที่จะลดขนาดของข้อมูลได้ 20% ถึง 90%

Huffman code

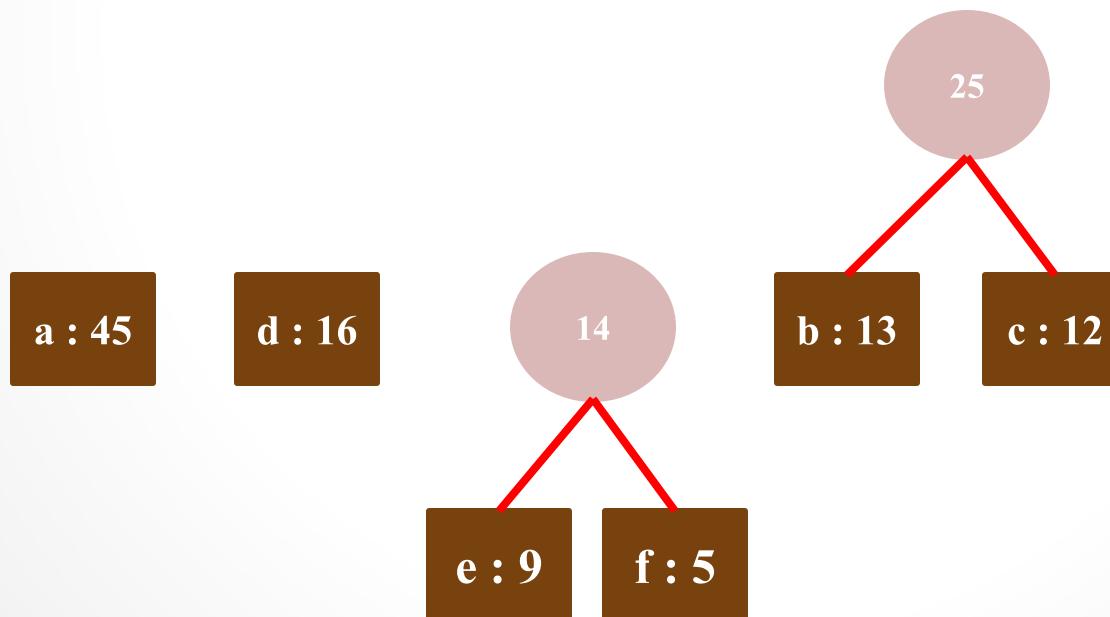
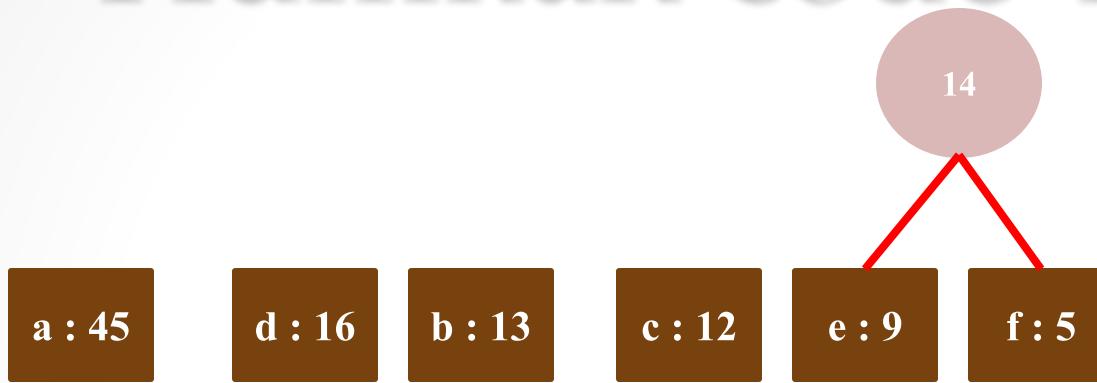
- ตัวอย่างชุดข้อมูล

	a	b	c	d	e	f
Frequency	45	13	12	16	9	5
Fix-Length	000	001	010	011	100	101
Variable-length	0	101	100	111	1101	1100

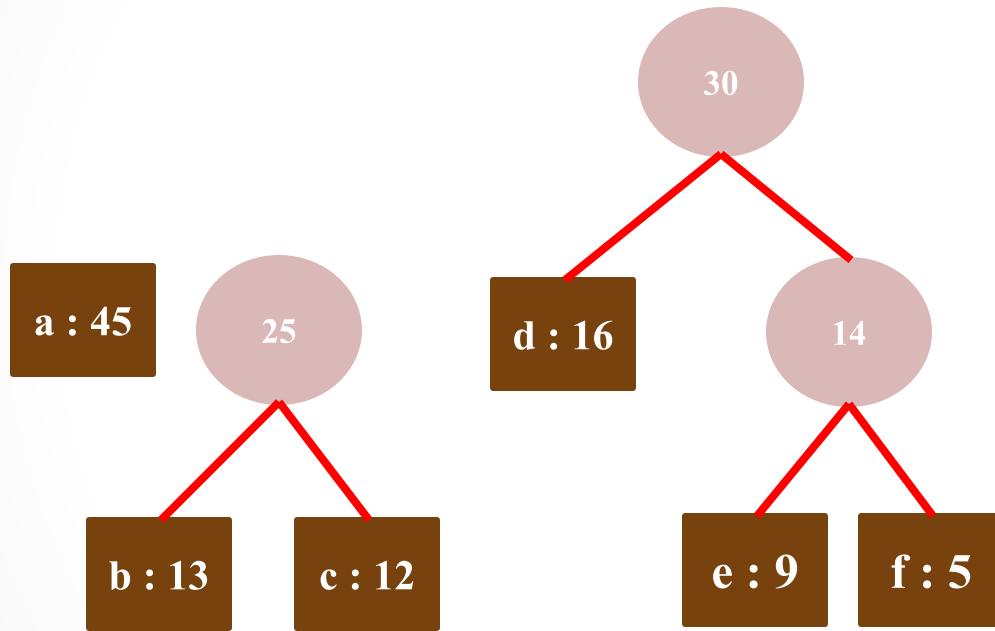
Fixed - Length



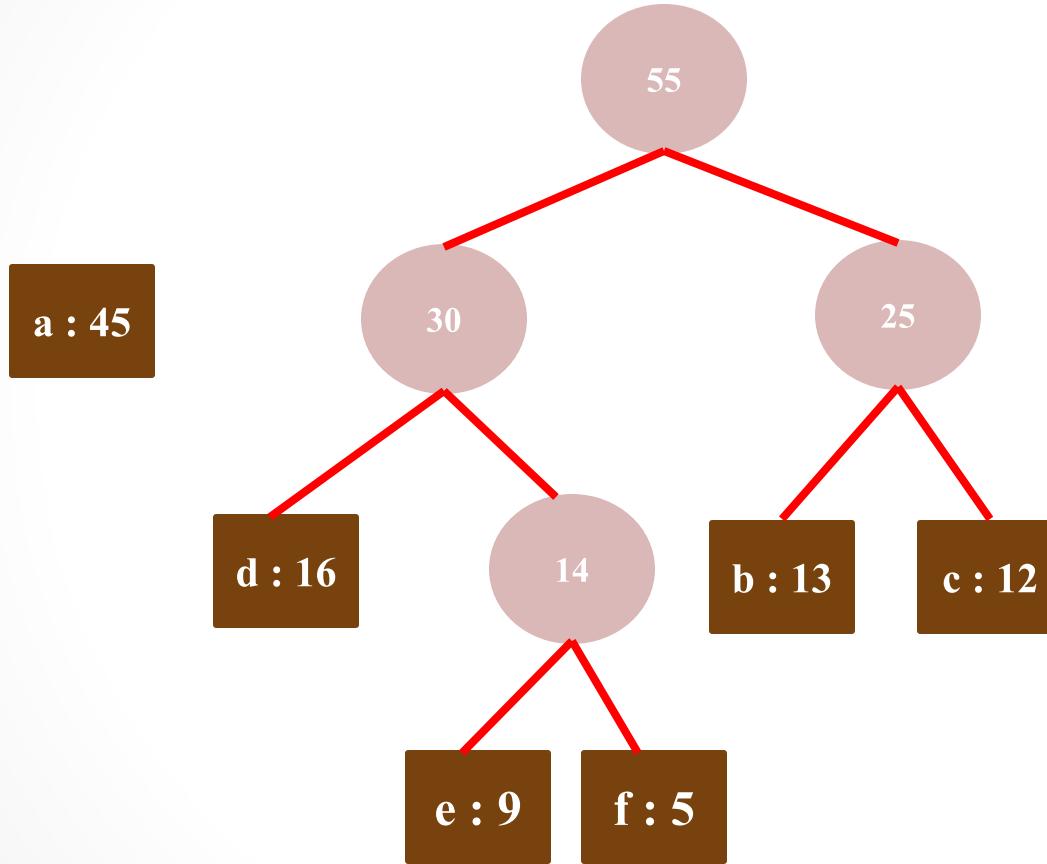
Variable – Length : Huffman code Tree



Variable – Length : Huffman code Tree



Variable – Length : Huffman code Tree



Variable – Length : Huffman code Tree

bad = 0101000

a แทนด้วยรหัส 1

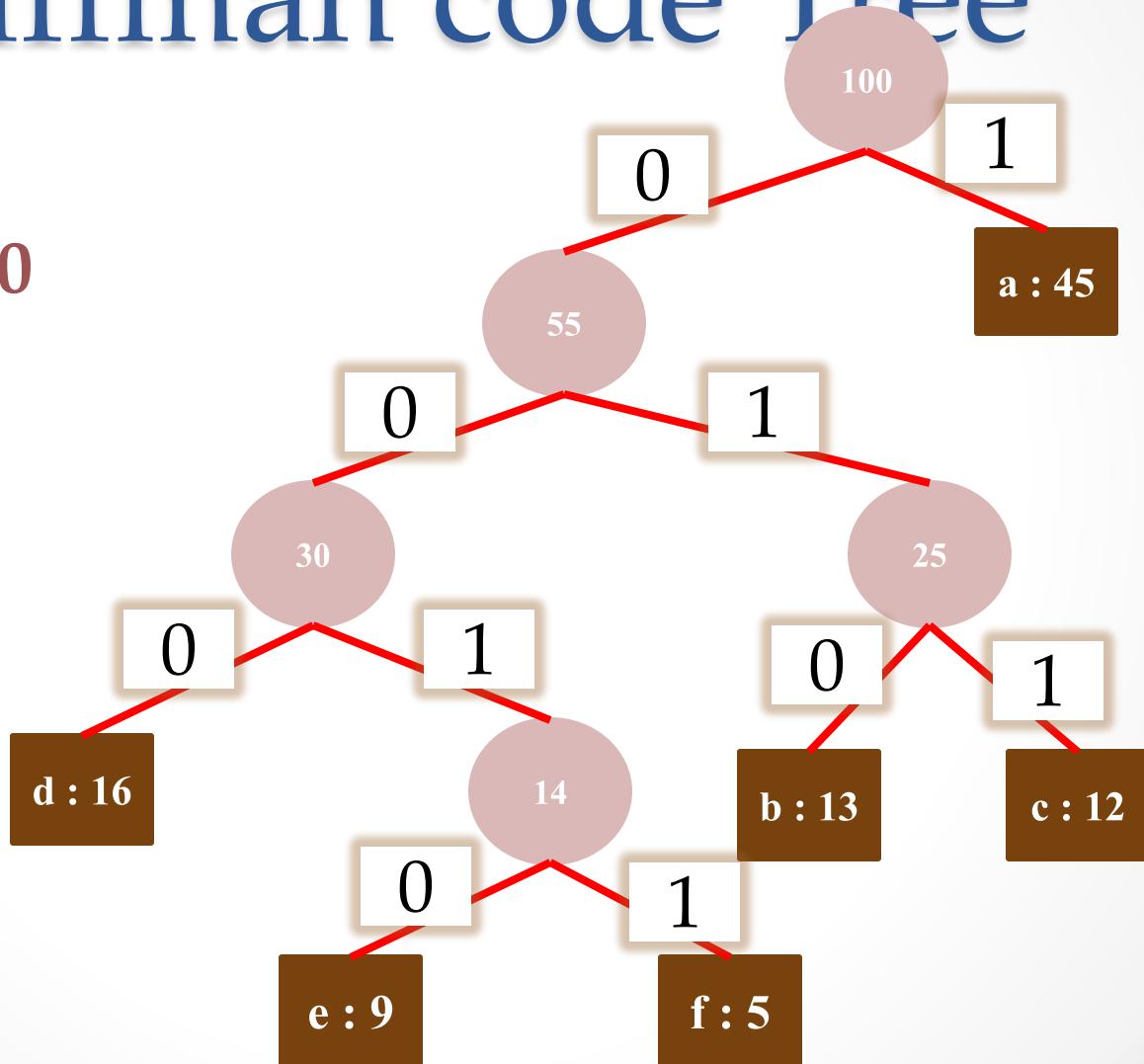
b แทนด้วยรหัส 010

c แทนด้วยรหัส 011

d แทนด้วยรหัส 000

e แทนด้วยรหัส 0010

f แทนด้วยรหัส 0011



Huffman code

- เป็นรหัสแทนตัวอักษรที่แต่ละตัวอักษรมีความยาวของรหัสแตกต่างกันโดย
- ตัวอักษรที่**ใช้บ่อย** จะมี**ขนาดสั้น**
- ตัวอักษรที่**ใช้น้อย** จะมี**ขนาดยาว**
- ทั้งนี้เพื่อทำให้ข้อมูลมีขนาดเล็กลง

แบบฝึกหัด

กำหนดความถี่ของการใช้งานตัวอักษรต่าง ๆ ดังนี้

$$A = 30\%$$

$$C = 15\%$$

$$E = 25\%$$

$$B = 15\%$$

$$D = 10\%$$

$$F = 5\%$$

1. จงสร้าง Huffman Code แทนตัวอักษรดังกล่าว
2. เขียนโปรแกรมเพื่อแสดง Huffman Code แทนตัวอักษรดังกล่าว