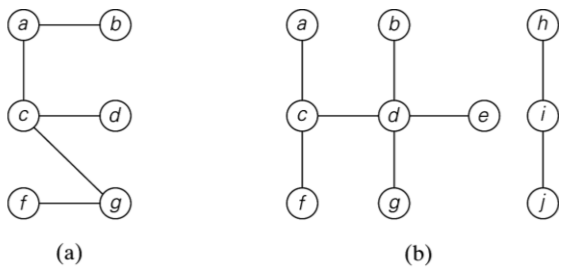
Tree (ต้นไม้)

1. นิยามและสมบัติของต้นไม้

ต้นไม้ (Tree) หมายถึง กราฟต่อเนื่อง (Connected) ที่ไม่มีวัฏจักร (Acyclic)

ป่า (Forest) หมายถึง กราฟที่ไม่มีวัฏจักร แต่ไม่จำเป็นต้องต่อเนื่อง



ภาพประกอบ 1.1 ต้นไม้ (a) และ ป่า (b)

<u>สมบัติของต้นไม้</u>

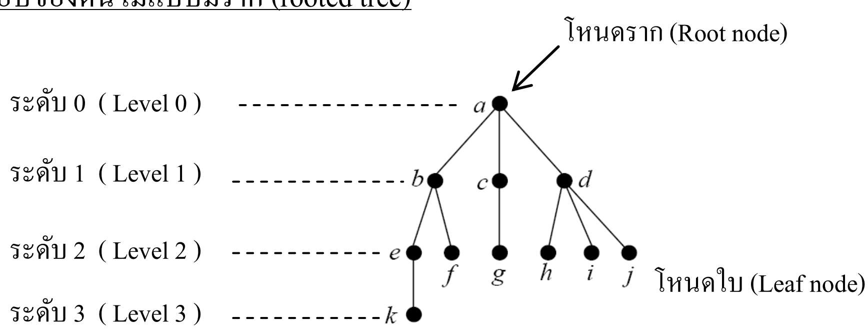
- มีเส้นทาง<u>เพียงเส้นทางเดียว</u>จากโหนดหนึ่ง ไปยังอีกโหนดหนึ่ง
- $|E| = |V| - 1$ เมื่อ $|E|$ คือ จำนวนเส้นเชื่อมในต้นไม้, $|V|$ คือ จำนวนโหนดในต้นไม้

2. ต้นไม้แบบมีราก (rooted tree)

(a) Free tree (b) Rooted tree

ภาพประกอบ 2.1 free tree vs. rooted tree

<u>ส่วนประกอบของต้นไม้แบบมีราก (rooted tree)</u>



ภาพประกอบ 2.2 ต้นไม้แบบมีราก (rooted tree)

1) *โหนดราก* (Root node) คือ โหนดแรกสุดของโครงสร้างต้นไม้ โหนดนี้จะเป็นโหนดเริ่มต้น จึงไม่มีโหนดอื่นที่อยู่ก่อนหน้ามัน จากภาพ a คือ โหนดราก

2) *ระดับของโหนด* (Node level) คือระยะทางตามแนวดิ่งของโหนดนั้นจากโหนดราก
   - กำหนดให้โหนดรากมีระดับเป็น 0
   - ระดับของโหนดถัดไปจากโหนดราก คือ 1 ถัดไปอีกระดับ คือ ระดับ 2 เรื่อยไปจนถึงระดับ n

3) *โหนดพ่อแม่* (Parent node) คือ โหนดที่มีระดับอยู่ก่อนหน้าหนึ่งระดับ จากภาพแสดงว่า
   - โหนด a เป็นพ่อแม่ของ b, c และ d
   - โหนด b เป็นพ่อแม่ของ e และ f
   - โหนด c เป็นพ่อแม่ของ g เป็นต้น

4) *โหนดลูก* (Child node) คือ โหนดที่ตามหลังในระดับถัดไป จากภาพแสดงว่า
   - โหนด b, c และ d เป็นลูกของ a
   - โหนด e และ f เป็นลูกของ b
   - โหนด g เป็นลูกของ c เป็นต้น

5) *โหนดบรรพบุรุษ* (Predecessor/Ancestor) คือ โหนดทุกโหนดที่อยู่บนเส้นทางจากโหนดรากของต้นไม้มายังโหนด นั้นๆ จากภาพแสดงว่า
   - โหนด a และ d เป็นบรรพบุรุษของ h
   - โหนด a, b และ e เป็นบรรพบุรุษของ k เป็นต้น

6) *โหนดลูกหลาน* (Successor/Descendant) คือ โหนดที่อยู่ตามหลัง หรือโหนดทุกโหนดที่มีโหนดนั้น ๆ เป็นบรรพบุรุษ จากภาพแสดงว่า
   - โหนด b, c, d, e, f, g, h, i, j และ k เป็นลูกหลานของ a
   - โหนด e และ k เป็นลูกหลานของ b
   - โหนด k เป็นลูกหลานของ e เป็นต้น

7) *โหนดพี่น้อง* (Siblings) คือ โหนดที่อยู่ในระดับเดียวกัน และมีพ่อแม่เดียวกัน จากภาพแสดงว่า
   - โหนด b, c และ d เป็นพี่น้องกัน
   - โหนด e และ f เป็นพี่น้องกัน เป็นต้น

8) *โหนดใบ* (Leaf node) คือ โหนดที่ไม่มีโหนดถัดไปหรือโหนดที่ไม่ลูกอยู่เลย จากภาพโหนด f, g, h, i, j และ k เป็นโหนดใบ

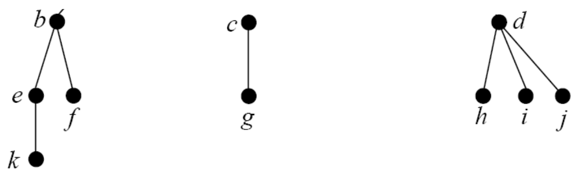
- โหนดใบ (Leaf node) เรียกอีกชื่อหนึ่งว่า *โหนดภายนอก* (External node) และเรียกโหนดอื่น ๆ ที่ไม่ใช่โหนดใบว่า *โหนดภายใน* (Internal node)

9) *ดีกรี* (Degree) คือ จำนวนลูก (Child) ทั้งหมดของโหนดนั้น จากภาพแสดงว่า

- โหนด a และ d มีดีกรี 3
- โหนด b มีดีกรี 2
- โหนด c และ e มีดีกรี 1
- โหนด $f, g, h, i, j,$ และ k มีดีกรี 0

10) *ต้นไม้ย่อย* (Subtree) ประกอบด้วยโหนดทุกโหนดที่เป็นลูกหลาน(Descendant) ของโหนดดังกล่าว

- ต้นไม้ย่อยของโหนด a



- ต้นไม้ย่อยของโหนด b

ต้นไม้แบบอันดับ (Ordered tree) คือ ต้นไม้แบบมีราก (Rooted tree) ที่มีการจัดอันดับของลูก ๆ ในแต่ละโหนดของต้นไม้ การสลับอันดับของลูก ๆ จะได้ต้นไม้ที่มีความหมายต่างกันไป

ต้นไม้ m ภาค (m-ary tree) คือ ต้นไม้แบบอันดับที่เรากำหนดได้แน่ ๆ ว่าจำนวนลูกของโหนดภายในจะมีไม่เกิน m ลูก

กรณีพิเศษเมื่อ $m = 2$ นั้นพบบ่อยที่สุดก็ว่าได้ในวงการคอมพิวเตอร์ จึงมีชื่อให้พิเศษว่าต้นไม้ทวิภาค (binary tree) นอกจากนี้ยังพิเศษขึ้นไปอีกตรงที่ว่าเราเรียกลูก ๆ ของโหนด ๆ หนึ่งในต้นไม้แบบทวิภาคว่าลูกทางซ้าย และลูกทางขวา (ไม่ใช่ลูกคนที่ 1 และลูกคนที่ 2)

3. ต้นไม้ทวิภาค (binary tree)

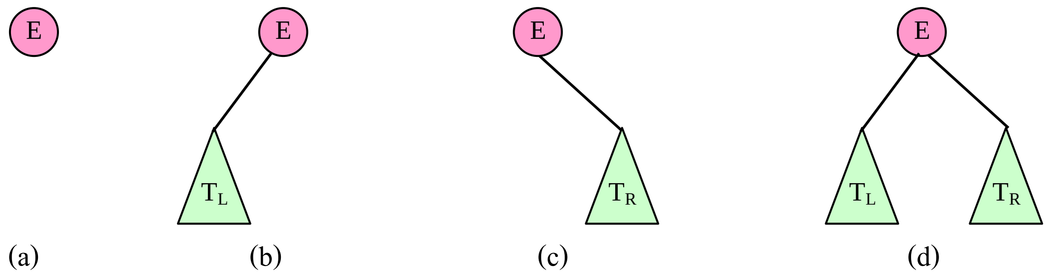
นอกจากการนิยามต้นไม้ทวิภาคตามลักษณะของต้นไม้ m ภาค เมื่อ m มีค่าเป็น 2 แล้ว เรายังสามารถนิยามต้นไม้ทวิภาคในลักษณะของการนิยามแบบเวียนบังเกิด (Recursive definition) ได้ โดยถ้าให้ T แทนต้นไม้ทวิภาคใด ๆ แล้ว T ได้แก่ เซตของโหนดของต้นไม้ทวิภาคที่มีสมบัติอย่างใดอย่างหนึ่งในสองกรณีต่อไปนี้

<u>กรณีที่ 1</u> เซต T เป็นเซตว่าง

<u>กรณีที่ 2</u> เซต T ประกอบด้วยเซตที่ไม่มีส่วนร่วม (disjoint set) จำนวน 3 เซต ซึ่งมีสมบัติ คือ

- เซตที่ประกอบด้วยสมาชิก 1 ตัว ได้แก่ *โหนดราก*
- เซตของต้นไม้ทวิภาคอีก 2 เซต ซึ่งเป็นต้นไม้ย่อยของโหนดราก เราเรียกต้นไม้ย่อยทั้งสองว่า *ต้นไม้ย่อยทางซ้าย (left subtree, T_L)* และ*ต้นไม้ย่อยทางขวา (right subtree, T_R)* ของโหนดราก

จะเห็นได้ว่า ถ้า T เป็นเซตว่างแล้ว T จะเป็นต้นไม้ทวิภาคว่าง ส่วนต้นไม้ทวิภาค กรณีที่ T ไม่เป็นเซตว่าง แสดงดังรูปต่อไปนี้



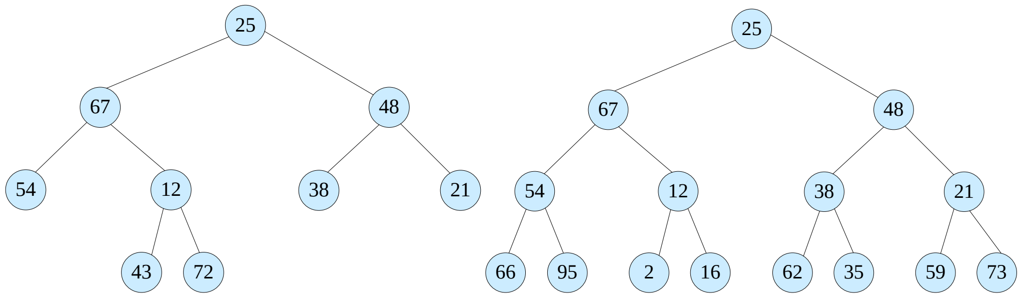
ภาพประกอบ 3.1 ต้นไม้ทวิภาคกรณีต่าง ๆ

- (a) กรณีต้นไม้ย่อยทางซ้าย (T_L) และต้นไม้ย่อยทางขวา (T_R) ของโหนดรากเป็นต้นไม้ว่าง
- (b), (c) และ (d) กรณีต้นไม้ย่อยทางซ้าย และ/หรือ ต้นไม้ย่อยทางขวาของโหนดรากไม่เป็นต้นไม้ว่าง

3.1 ประเภทของต้นไม้ทวิภาค

ต้นไม้ทวิภาคแบบเต็ม (Full binary tree) ได้แก่ ต้นไม้ทวิภาคที่โหนดภายในทุกโหนดมีดีกรีเท่ากับ 2

ต้นไม้ทวิภาคแบบสมบูรณ์ (Complete binary tree)ได้แก่ ต้นไม้ทวิภาคที่โหนดใบทุกโหนดอยู่ในระดับเดียวกัน และโหนดภายในทุกโหนดมีดีกรีเท่ากับ 2

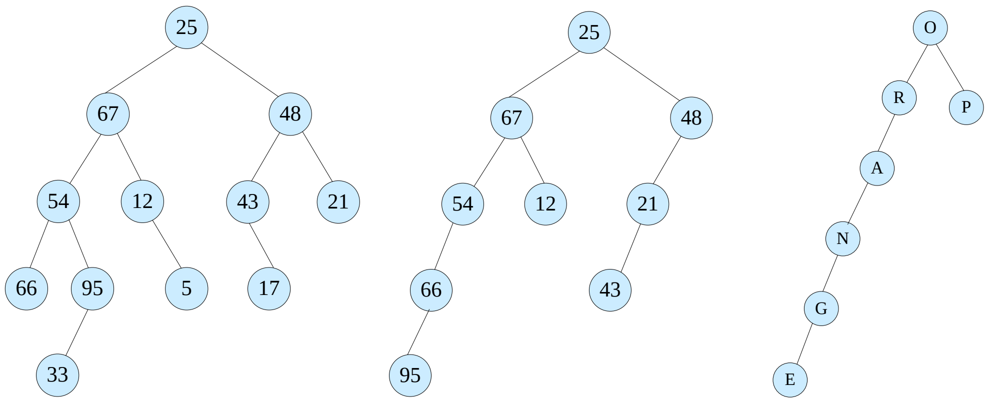


Full binary tree Complete binary tree

ภาพประกอบ 3.2

ต้นไม้ทวิภาคแบบสมดุล (Balance binary tree) ได้แก่ ต้นไม้ทวิภาคที่มีความสูงของต้นไม้ย่อยทางขวาของโหนดใด ๆ ต่างจากความสูงของต้นไม้ย่อยทางซ้ายของโหนดเดียวกัน ไม่เกิน 1

นั่นคือ เมื่อ T เป็นโหนดใด ๆ ในต้นไม้ทวิภาค แล้วต้นไม้ทวิภาคนี้จะเป็นต้นไม้ทวิภาคสมดุล ก็ต่อเมื่อ $\left|height(\mathbf{T}_L) - height(\mathbf{T}_R)\right| \le 1$



(a) (b) (c)

ภาพประกอบ 3.3

(a) เป็นต้นไม้ทวิภาคแบบสมดุล (b) และ (c) เป็นต้นไม้ทวิภาคแบบไม่สมดุล

3.2 โครงสร้างข้อมูลของต้นไม้ทวิภาค

- โครงสร้างข้อมูลต้นไม้ทวิภาคแบบหน่วยเก็บต่อเนื่อง (continuous storage)
- โครงสร้างข้อมูลต้นไม้ทวิภาคแบบแถวลำดับ (array-based)
- โครงสร้างข้อมูลต้นไม้ทวิภาคแบบตัวชี้ (pointer-based)

3.2.1 โครงสร้างข้อมูลต้นไม้ทวิภาคแบบหน่วยเก็บต่อเนื่อง (continuous storage)

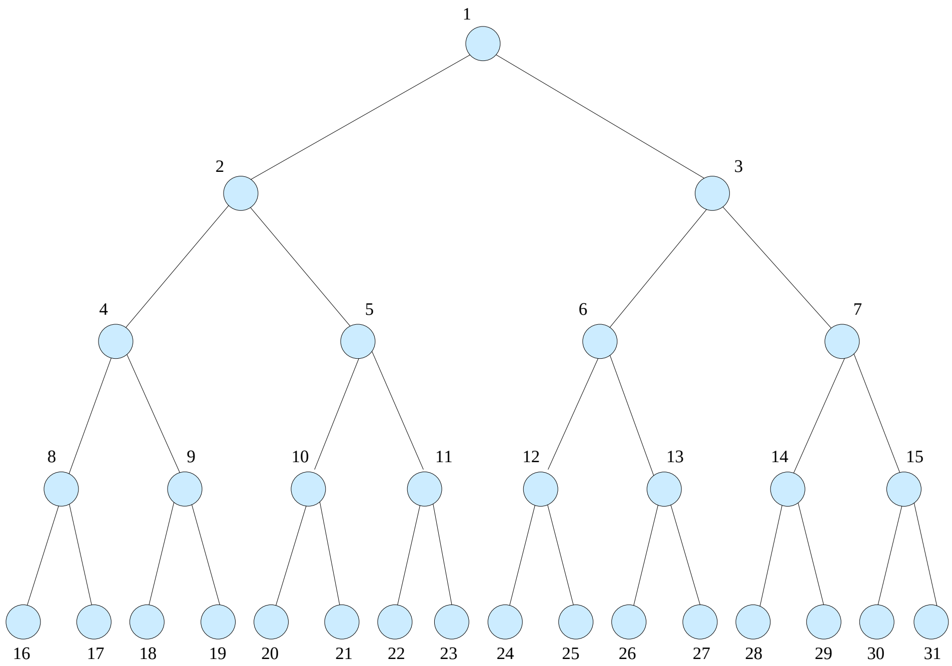
ใช้แถวลำดับ 1 มิติ สำหรับจัดเก็บข้อมูลโหนดต่าง ๆ ของต้นไม้ทวิภาค โดยกำหนดให้

แถวลำดับตำแหน่งที่ 1 เก็บโหนดราก

แถวลำดับตำแหน่งที่ 2 เก็บโหนดลูกทางซ้ายของโหนดราก

แถวลำดับตำแหน่งที่ 3 เก็บโหนดลูกทางขวาของโหนดราก

....



ภาพประกอบ 3.4 ตำแหน่งของแถวลำดับในต้นไม้ทวิภาคแบบหน่วยเก็บต่อเนื่อง

จากภาพประกอบ 3.4 เราสามารถคำนวณตำแหน่งของโหนดในต้นไม้ทวิภาคจากตำแหน่งในแถวลำดับได้ ดังนี้

- ตำแหน่งโหนดลูกทางซ้ายของโหนดในแถวลำดับตำแหน่ง k คือ ตำแหน่งที่ 2k ในแถวลำดับ
- ตำแหน่งโหนดลูกทางขวาของโหนดในแถวลำดับตำแหน่ง k คือ ตำแหน่งที่ 2k+1 ในแถวลำดับ
- ตำแหน่งโหนดพ่อแม่ของโหนดในแถวลำดับตำแหน่งที่ k คือ ตำแหน่งที่ $\lfloor k/2 \rfloor$ ในแถวลำดับ

<u>หมายเหตุ</u> $\lfloor x \rfloor$ คือ จำนวนเต็มที่มีค่าสูงที่สุดที่น้อยกว่า x

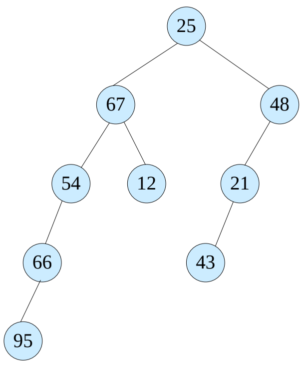
<u>ตัวอย่างเช่น</u>

- ตำแหน่งโหนดลูกทางซ้ายและโหนดลูกทางขวาของโหนดในแถวลำดับตำแหน่งที่ 5 คือ ตำแหน่งที่ 10 และ 11 ในแถวลำดับ
- ตำแหน่งโหนดพ่อแม่ของโหนดในแถวลำดับตำแหน่งที่ 27 คือ ตำแหน่งที่ $\lfloor 27/2 \rfloor = 13$ ในแถวลำดับ

การประกาศโครงสร้างข้อมูลแบบหน่วยเก็บต่อเนื่องสำหรับต้นไม้ทวิภาคที่มีจำนวนโหนดไม่เกิน 31 โหนด สามารถทำได้ดังนี้

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

เนื่องจากแถวลำดับในตำแหน่งที่ 0 ไม่ได้ใช้เก็บข้อมูลของโหนด เราจึงสามารถใช้ตำแหน่งดังกล่าวเก็บจำนวนโหนดที่มีข้อมูลของต้นไม้ ดังรูปข้างล่างนี้



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

ต้นไม้ทวิภาค `t` โครงสร้างแบบหน่วยเก็บต่อเนื่องของต้นไม้ทวิภาค `t`

ภาพประกอบ 3.5 โครงสร้างแบบหน่วยเก็บต่อเนื่องของต้นไม้ทวิภาค

ข้อดีและข้อเสียของโครงสร้างข้อมูลต้นไม้ทวิภาคแบบหน่วยเก็บต่อเนื่อง

<u>ข้อดี</u>
- สะดวกต่อการเขียนโปรแกรม
- การคำนวณหาตำแหน่งโหนดลูกทางซ้ายและขวา รวมทั้งโหนดพ่อแม่ สามารถทำได้อย่างรวดเร็ว เนื่องจาก ทุกโหนดในต้นไม้มีตำแหน่งที่แน่นอนในแถวลำดับ

<u>ข้อเสีย</u>
- สิ้นเปลืองพื้นที่ในแถวลำดับ ในกรณีที่โหนดของต้นไม้ทวิภาคส่วนใหญ่เป็นโหนดว่าง

3.2.2 โครงสร้างข้อมูลต้นไม้ทวิภาคแบบแถวลำดับ (array-based)

- ใช้แถวลำดับ 1 มิติ ในการเก็บโหนดต่าง ๆ ในต้นไม้ทวิภาคเช่นกัน
- ไม่มีการกำหนดตำแหน่งที่แน่นอนให้กับโหนดใด ๆ รวมทั้งโหนดราก ในต้นไม้ทวิภาค
- เก็บตำแหน่งของโหนดราก ตำแหน่งโหนดลูกทางซ้ายและตำแหน่งโหนดลูกทางขวาของแต่ละโหนด เพื่อให้สามารถทราบโครงสร้างของโหนดภายในต้นไม้ทวิภาคได้

การประกาศโครงสร้างข้อมูลแบบแถวลำดับสำหรับต้นไม้ทวิภาคที่มีจำนวน โหนดไม่เกิน 31 โหนด สามารถทำได้ดังนี้

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

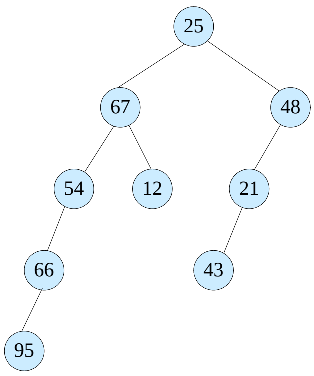
จากโครงสร้างนี้ `t.table[MAXNODES]` เป็นแถวลำดับ 1 มิติ ขนาด `MAXNODES` ใช้จัดเก็บลิสต์ 2 ลิสต์ด้วยกัน คือ

- ลิสต์ของโหนดที่มีข้อมูลในต้นไม้ทวิภาค ซึ่งมี `t.rootIndex` เป็นตำแหน่งแรกของลิสต์
- ลิสต์ของโหนดว่างในต้นไม้ทวิภาค ซึ่งมี `t.freeListIndex` เป็นตำแหน่งแรกของลิสต์

เมื่อ `t.rootIndex` และ `t.freeListIndex` มีค่าเป็น -1 แสดงว่าลิสต์ของโหนดที่มีข้อมูลและลิสต์ของโหนดว่าง เป็นลิสต์ว่าง ตามลำดับ แต่เมื่อ `t.rootIndex` มีค่าเป็น `i` โดยที่ $0 \le i \le 30$ เราจะได้ว่า `i` เป็นตำแหน่งของโหนดรากของต้นไม้ทวิภาค `t`

`t.table[i].data` ใช้สำหรับเก็บข้อมูลของโหนด `i` ส่วน `t.table[i].left` และ `t.table[i].right` ใช้สำหรับเก็บตำแหน่งโหนดลูกทางซ้ายและตำแหน่งโหนดลูกทางขวาของโหนด `i`

ถ้า `t.table[i].left` และ `t.table[i].right` มีค่าเป็น -1 แสดงว่าโหนด `i` ไม่มีโหนดลูกทางซ้าย และโหนดลูกทางขวา ตามลำดับ



ต้นไม้ทวิภาค `t`

binaryTree t

rootIndex: -1

freeListIndex: 0

table

node

[]	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

ภาพประกอบ 3.6 โครงสร้างแบบแถวลำดับในต้นไม้ทวิภาคในสถานะต้นไม้ว่าง

ภาพประกอบ 3.6 แสดงโครงสร้างข้อมูลแบบแถวลำดับของต้นไม้ทวิภาค t โดยเมื่อเริ่มต้นทำงาน ต้นไม้ทวิภาค t มีสถานะเป็นต้นไม้ว่าง ลิสต์ของโหนดที่มีข้อมูลเป็นลิสต์ว่าง นั่นคือ `t.rootIndex` มีค่าเท่ากับ `-1` และลิสต์ของโหนดว่างเป็นลิสต์ไม่ว่าง โดยโหนดแรกของลิสต์ของโหนดว่า ได้แก่ โหนดตำแหน่งที่ 0 ดังนั้น `t.freeListIndex` จึงมีค่าเท่ากับ 0 โหนดแต่ละโหนดในลิสต์นี้เชื่อมโยงกันผ่านค่า `right` (หรืออาจเชื่อมโยงผ่านค่า `left` แทนได้)

binaryTree t

rootIndex 0

freeListIndex 1

table

[]	left	data	right
0	-1	25	-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

binaryTree t

rootIndex 0

freeListIndex 2

table

[]	left	data	right
0	1	25	-1
1	-1	67	-1
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

binaryTree t

rootIndex 0

freeListIndex 3

table

[]	left	data	right
0	1	25	2
1	-1	67	-1
2	-1	48	-1
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

เพิ่มโหนด 25 เป็นโหนดรากของต้นไม้ทวิภาค t เพิ่มโหนด 67 เป็นโหนดลูกทางซ้ายของโหนดราก เพิ่มโหนด 48 เป็นโหนดลูกทางขวาของโหนดราก

binaryTree t

rootIndex 0

freeListIndex 4

table

[]	left	data	right
0	1	25	2
1	-1	67	-1
2	3	48	-1
3	-1	21	-1
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

binaryTree t

rootIndex 0

freeListIndex 5

table

[]	left	data	right
0	1	25	2
1	4	67	-1
2	3	48	-1
3	-1	21	-1
4	-1	54	-1
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

binaryTree t

rootIndex 0

freeListIndex 6

table

[]	left	data	right
0	1	25	2
1	4	67	5
2	3	48	-1
3	-1	21	-1
4	-1	54	-1
5	-1	12	-1
6	-1	0	7
7	-1	0	8
8	-1	0	9
9	-1	0	10
.	.	.	.
.	.	.	.
29	-1	0	30
30	-1	0	-1

เพิ่มโหนด 21 เป็นโหนดลูกทางซ้ายของโหนด 48 เพิ่มโหนด 54 เป็นโหนดลูกทางซ้ายของโหนด 67 เพิ่มโหนด 12 เป็นโหนดลูกทางขวาของโหนด 67

ภาพประกอบ 3.7 ต้นไม้ทวิภาค t หลังจากเพิ่มโหนด 25, 67, 48, 21, 54, และ 12 ตามลำดับ

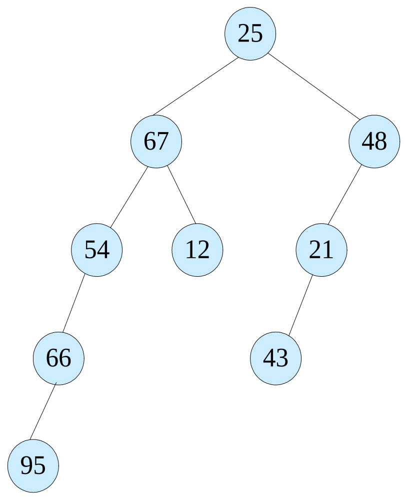
`binaryTree t`

`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



เพิ่มโหนด 43 เป็นโหนดลูกทางซ้ายของโหนด 21
เพิ่มโหนด 66 เป็นโหนดลูกทางซ้ายของโหนด 54
เพิ่มโหนด 95 เป็นโหนดลูกทางซ้ายของโหนด 66

ภาพประกอบ 3.8 ต้นไม้ทวิภาค t หลังจากเพิ่มโหนดทั้ง 9 โหนด

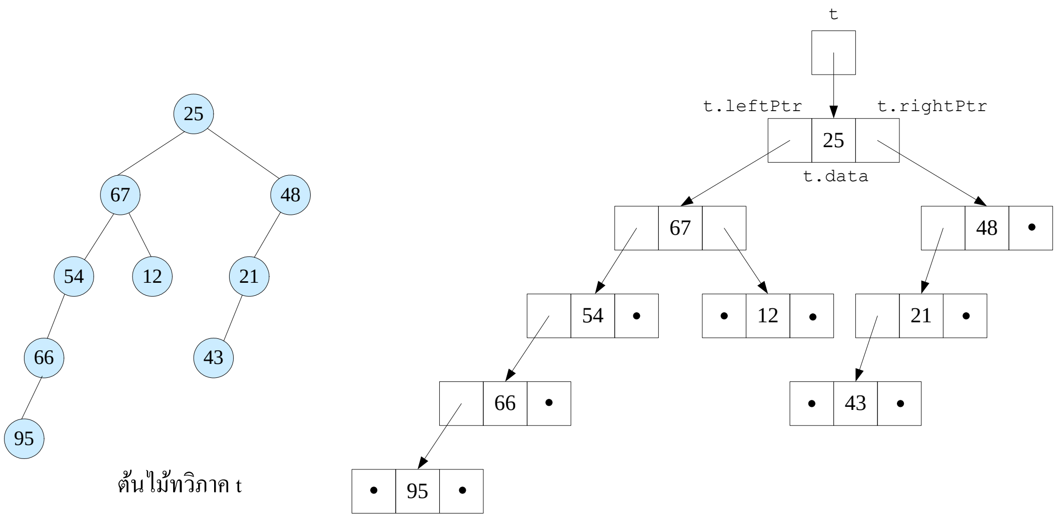
3.2.3 โครงสร้างข้อมูลต้นไม้ทวิภาคแบบตัวชี้ (pointer-based)

จำนวนโหนดทั้งหมดของต้นไม้ทวิภาคมีการเปลี่ยนแปลงไปตามการทำงานที่เกิดขึ้นจริง

```
struct node {
      struct node *leftPtr;
      int         data;
      struct node *rightPtr;
};

typedef struct node BTNode;
typedef BTNode *binaryTree;

binaryTree t;
```



ภาพประกอบ 3.9 โครงสร้างแบบตัวชี้ของต้นไม้ทวิภาค t

เช่นเดียวกันกับโครงสร้างข้อมูลต้นไม้ทวิภาคแบบแถวลำดับ โครงสร้างข้อมูลต้นไม้ทวิภาคแบบตัวชี้ไม่กำหนดตำแหน่งที่แน่นอนของโหนด เพียงแต่ตำแหน่งของโหนดในกรณีหลังกำหนดผ่านตัวชี้ ซึ่งได้แก่ตำแหน่งของโหนดบนหน่วยความจำ ขณะที่ในกรณีแรกกำหนดผ่านตำแหน่งภายในแถวลำดับ

3.3 การดำเนินการต่าง ๆ กับ ต้นไม้ทวิภาค

การดำเนินการกับต้นไม้จะมีอะไรบ้างนั้น ทั้งนี้ก็ขึ้นกับว่า เราประยุกต์ต้นไม้กับงานอะไร และต้องการบริการใดจากการจัดเก็บดังกล่าว ในหัวข้อนี้จะขอนำเสนอตัวอย่างการดำเนินการทั่วไป พอให้เห็นเป็นแนวทางการเขียน procedure หรือ function ที่กระทำกับต้นไม้ ดังนี้

- การนับจำนวนโหนด
- การหาความสูง
- การแวะผ่านต้นไม้

3.3.1 การนับจำนวนโหนด

ตัวการการดำเนินการที่จะนำเสนอต่อไปนี้ อาศัยธรรมชาติแบบเวียนเกิดของโครงสร้างของต้นไม้ (recursive structure) หมายความว่าเราสามารถมองต้นไม้แบบทวิภาคว่าเป็นต้นไม้ที่ประกอบด้วยราก และต้นไม้แบบทวิภาคย่อยสองต้นซึ่งถูกนำมาต่อเป็นลูกของรากนั้น นั่นคือการนิยามต้นไม้แบบทวิภาคด้วยต้นไม้แบบทวิภาค

กำหนดให้ $T(r)$ คือ ต้นไม้แบบทวิภาคที่มีโหนด r เป็นราก

$N(T(r))$ คือ จำนวนโหนดของต้นไม้ $T(r)$

$L(r)$ คือ โหนดลูกซ้ายของ r

$R(r)$ คือ โหนดลูกขวาของ r

เราสามารถเขียนนิยามของจำนวนโหนดในต้นไม้ $T(r)$ ได้ดังนี้

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

ซึ่งให้ความหมายว่าถ้าเป็นต้นไม้ว่าง (รากเป็น nil) จำนวนโหนดก็เป็น 0 ถ้ามีโหนดราก จำนวนโหนดของทั้งต้นก็ย่อมเท่ากับ 1 (ซึ่งคือการนับโหนดราก) บวกกับจำนวนโหนดของต้นไม้ย่อยทางซ้าย และจำนวนโหนดของต้นไม้ย่อยทางขวา เราสามารถเขียนเป็นฟังก์ชันได้ตรงไปตรงมาจากนิยามข้างต้นดังนี้

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

3.3.2 การหาความสูง

จากนิยามของความสูงของต้นไม้ซึ่งคือความยาวของวิถีจากรากถึงใบที่ลูกที่สุดในต้นไม้ แล้วเราหาใบที่ลึกที่สุดอย่างไร ลองจินตนการดูว่าถ้าเราอยู่ที่โหนดๆ หนึ่งในต้นไม้ เราจะทราบได้อย่างไรว่าใบที่ลึกที่สุดในต้นไม้นั้นเป็นลูกหลานทางซ้ายหรือทางขวาจากโหนดที่เราอยู่ คำตอบก็คือ เราไม่ทราบจนกว่าจะได้ลองไปดูทั้งสองทางทั้งทางซ้ายและขวา แล้วจึงนำมาเปรียบเทียบกัน และแทนที่เราจะหาใบลึกที่สุด ถ้าเราหันมานึกถึงโครงสร้างแบบเวียนเกิดของต้นไม้ จะพบว่า **ความสูงของโหนดใด ย่อมคิดมาจากความสูงของต้นไม้ย่อยต้นที่สูงกว่าระหว่างต้นไม้ย่อยทั้งสองซึ่งเป็นลูกของโหนดนั้น**

กำหนดให้ $T(r)$ คือ ต้นไม้แบบทวิภาคที่มีโหนด r เป็นราก

$H(T(r))$ คือ จำนวนโหนดของต้นไม้ $T(r)$

$L(r)$ คือ โหนดลูกซ้ายของ r

$R(r)$ คือ โหนดลูกขวาของ r

เราสามารถเขียนนิยามของจำนวนโหนดในต้นไม้ $T(r)$ ได้ดังนี้

$$H(T(r)) = \begin{cases} 0 & \text{if} \quad L(r) = nil \text{ AND } R(r) = nil \\ 1 + H(T(L(r))) & \text{if} \quad L(r) \neq nil \text{ AND } R(r) = nil \\ 1 + H(T(R(r))) & \text{if} \quad L(r) = nil \text{ AND } R(r) \neq nil \\ 1 + \max\left(H(T(L(r))), H(T(R(r)))\right) & \text{if} \quad L(r) \neq nil \text{ AND } R(r) \neq nil \end{cases}$$

ซึ่งแบ่งวิธีการหาความสูงออกเป็นสี่กรณีคือ กรณีไม่มีลูกทั้งสอง ซึ่งหมายความว่าเป็นใบ มีความสูงเป็น 0 กรณีมีเฉพาะลูกซ้าย ความสูงย่อมเท่าความสูงของต้นซ้ายที่เป็นลูกบวกอีกหนึ่ง และในทำนองเดียวกัน กรณีมีเฉพาะลูกขวา ความสูงย่อมเท่าความสูงของต้นขวาที่เป็นลูกบวกอีกหนึ่ง แต่กรณีมีทั้งสองลูกก็ต้องนำความสูงของต้นไม้ย่อยที่เป็นลูกที่สูงกว่า บวกอีกหนึ่ง

แล้วถ้าเป็นกรณีต้นไม้ว่าง จะสูงเท่าใด ตรงนี้ขอให้คิดแบบนี้ ถ้ามีเพียงโหนดเดียวแสดงว่าสูงศูนย์ ดังนั้นจึงขอกำหนดว่าต้นไม้ว่างให้สูง –1 หากกำหนดเช่นนี้ ส่งผลให้เราสามารถเขียนนิยามของ $H(T(r))$ ได้กะทัดรัดขึ้นเหลือเพียงสองกรณี ดังนี้

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

จากนิยามข้างบนนี้เราสามารถเขียนเป็นฟังก์ชันได้อย่างตรงไปตรงมา ดังนี้

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

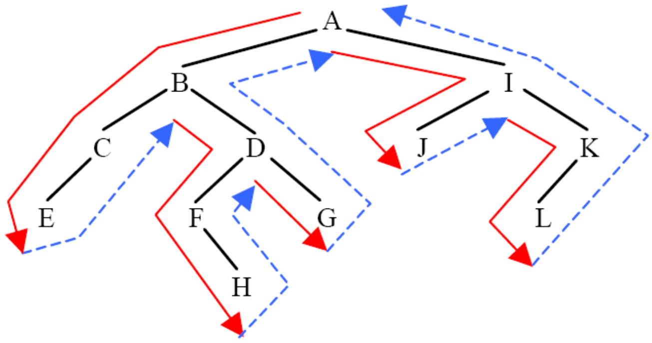
3.3.3 การแวะผ่านต้นไม้(tree traversal)

การดำเนินการบนต้นไม้ที่พบเห็นบ่อยมากคือการแวะผ่านต้นไม้ (tree traversal) ซึ่งเป็นขั้นตอนการวิ่งเข้าไปในต้นไม้โดยมีจุดประสงค์เพื่อวิ่งผ่านให้ครบทุกโหนด คำถามที่ตามมาก็คือจะแวะโหนดทุก ๆ โหนดไปทำไม ? คำตอบก็คงขึ้นกับลักษณะของการจัดเก็บข้อมูลในต้นไม้ว่าแทนอะไรอยู่ ตัวอย่างที่เห็นได้ชัดก็คือการแวะผ่านต้นไม้เพื่อพิมพ์ข้อมูลของทุก ๆ โหนดในต้นไม้ เป็นต้น เมื่อเราต้องการแวะผ่านต้นไม้ ก็มีคำถามอีกว่าจะแวะที่โหนดใดก่อน โหนดใดหลัง หมายความว่าลำดับของโหนดที่ถูกแวะนั้นจะเป็นอย่างไร ในหัวข้อจะได้นำเสนอวิธีการแวะผ่านต้นไม้ที่ใช้กันมาก โดยมีลำดับของโหนดที่ถูกแวะผ่านต่างๆ กัน สามวิธี คือ

- การแวะผ่านแบบก่อนลำดับ (preorder traversal)
- การแวะผ่านแบบตามลำดับ (inorder traversal)
- การแวะผ่านแบบหลังลำดับ (postorder traversal)

การแวะผ่านแบบก่อนลำดับ (preorder traversal)

การแวะผ่านแบบก่อนลำดับ (preorder traversal) นั้นเริ่มแวะรากของต้นไม้ก่อน จากนั้นวิ่งแวะโหนดลงไปตามแนวลึกลงไปเรื่อย ๆ วิ่งลงผ่านโหนดใดก็แวะโหนดนั้น โดยเลือกที่จะวิ่งลงทางซ้ายก่อนเสมอเมื่อใดวิ่งลงต่อไม่ได้ ก็วิ่งย้อนขึ้นตามทางเดิมที่ผ่านมา ผ่านโหนดใดที่ยังไม่เคยลงทางขวา ก็ลงลุยต่อตามแนวลึกในลักษณะที่กล่าวมา



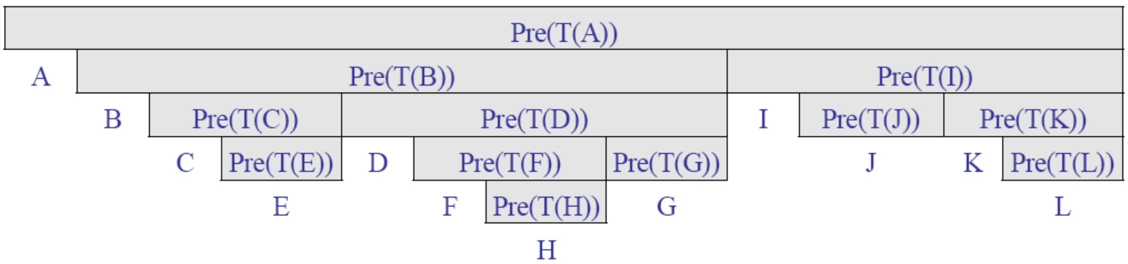
ภาพแสดงตัวอย่างการแวะผ่านต้นไม้แบบก่อนลำดับ เริ่มจากรากวิ่งลงทางซ้ายตามแนวที่แสดงด้วยเส้นทึบสีแดง ลงมาต้นที่ E จึงย้อนขึ้น ตามแนวเส้นประสีน้ำเงิน จนถึง B พบว่ามีทางแยกขวาก็ลงต่อตามแนวเส้นทึบสีแดง ต้นที่ H ก็วิ่งย้อนขึ้น และกระทำในลักษณะวิ่งลงลึก และมีการย้อนขึ้นเช่นนี้จนกระทั่งกลับมาที่รากของต้นไม้อีกครั้งหนึ่ง เมื่อใดที่ผ่านโหนดตามทิศที่กำลังวิ่งลง ก็แวะโหนดนั้นด้วย ดังนั้น ลำดับการแวะโหนดของต้นไม้ในภาพข้างต้น จึงเป็น A, B, C, E, D, F, H, G, I, J, K, L

แล้วจะเขียนเป็นโปรแกรมได้อย่างไร? การแวะผ่านแบบก่อนลำดับนั้นอาศัยแนวคิดของการทำงานแบบเวียนเกิด คือถ้าต้องการแวะผ่านต้นไม้แบบก่อนลำดับในต้นไม้ที่มีรากเป็น r ก็ให้แวะโหนด r ก่อน จากนั้นจึงตามด้วยการแวะผ่านต้นไม้ย่อยทางซ้ายของ r แบบก่อนลำดับ และตามด้วยการแวะผ่านต้นไม้ย่อยทางขวาของ r แบบก่อนลำดับ

กำหนดให้ $Pre(T(r))$ คือ ลำดับของโหนดที่ถูกแวะด้วยการแวะผ่านแบบก่อนลำดับในต้นไม้ที่มี r เป็นราก จะได้ว่า

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

ดังนั้นการแวะผ่านต้นไม้แบบก่อนลำดับในภาพข้างต้น จากนิยามข้างบนนี้แสดงได้ผังข้างล่างนี้ ผลที่ได้ก็คือลำดับ A, B, C, E, D, F, H, G, I, J, K, L



จากนิยามการแวะผ่านแบบก่อนลำดับข้างบนนี้สามารถเขียนเป็นโปรแกรมแบบเวียนเกิดได้ดังนี้

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

Visit ที่ปรากฏในโปรแกรมข้างบนนี้ แทนกระบวนการแวะโหนด ซึ่งจะทำอะไรนั้น ก็ขึ้นกับปัญหาที่สนใจ (เช่นถ้าต้องการพิมพ์ข้อมูลในทุก ๆ โหนด visit ก็อาจแทนได้ด้วย print เป็นต้น)

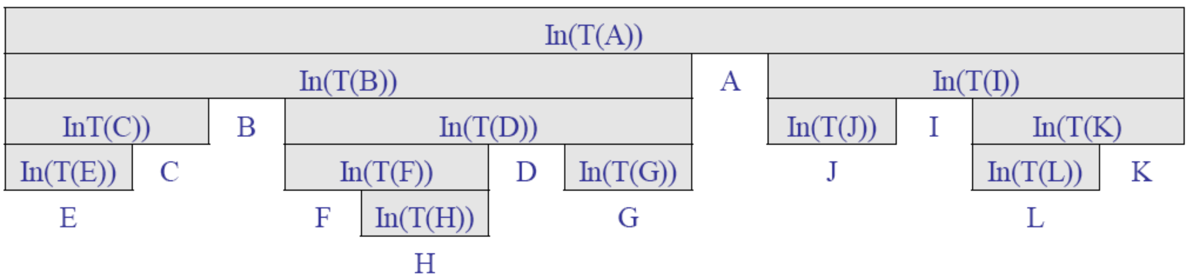
การแวะผ่านแบบตามลำดับ (inorder traversal)

การแวะผ่านแบบตามลำดับ (inorder traversal) ก็มีขั้นตอนการทำงานคล้ายกับการแวะผ่านแบบก่อนลำดับ จะต่างกันก็ตรงที่ลำดับที่เราแวะโหนดราก โดยการแวะผ่านต้นไม้แบบตามลำดับในต้นไม้ที่มีรากเป็น r <u>จะเริ่มด้วยการแวะผ่านต้นไม้ย่อยทางซ้ายของ r แบบตามลำดับก่อน ตามด้วยการแวะโหนดราก r แล้วจึงค่อยการแวะผ่านต้นไม้ย่อยทางขวาของ r แบบตามลำดับ</u>

กำหนดให้ $In(T(r))$ คือ ลำดับของโหนดที่ถูกแวะด้วยการแวะผ่านแบบตามลำดับในต้นไม้ที่มี r เป็นราก จะได้ว่า

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

ดังนั้นการแวะผ่านต้นไม้แบบตามลำดับในภาพข้างต้น จากนิยามข้างบนนี้แสดงได้ผังข้างล่างนี้ ผลที่ได้ก็คือลำดับ E, C, B, F, H, D, G, A, J, I, L, K



จากนิยามการแวะผ่านแบบตามลำดับข้างบนนี้สามารถเขียนเป็นโปรแกรมแบบเวียนเกิดได้ดังนี้

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

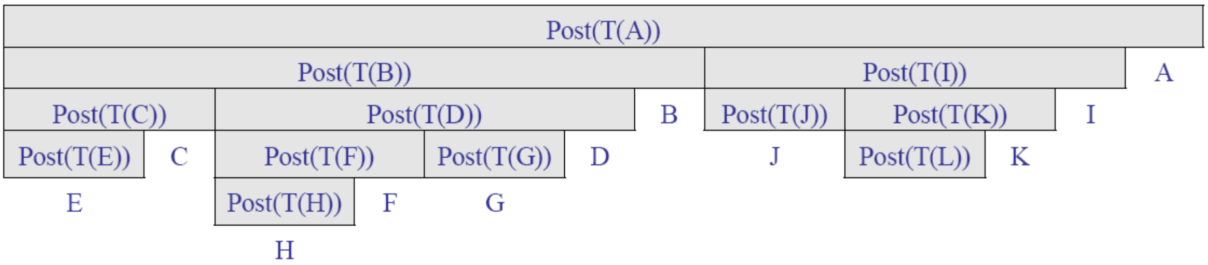
การแวะผ่านแบบหลังลำดับ (postorder traversal)

การแวะผ่านแบบหลังลำดับ (postorder traversal) ก็มีขั้นตอนการทำงานคล้ายกับการแวะผ่านทั้งสองแบบที่ได้นำเสนอมา จะต่างกันก็ตรงที่ลำดับที่เราแวะโหนดราก โดยการแวะผ่านต้นไม้แบบหลังลำดับในต้นไม้ที่มีรากเป็น r จะเริ่มด้วยการแวะผ่านต้นไม้ย่อยทางซ้ายของ r แบบหลังลำดับก่อน ตามด้วยการแวะผ่านต้นไม้ย่อยทางขวาของ r แบบหลังลำดับ แล้วจึงค่อยแวะโหนดราก r

กำหนดให้ $Post(T(r))$ คือลำดับของโหนดที่ถูกแวะด้วยการแวะผ่านแบบหลังลำดับในต้นไม้ที่มี r เป็นราก จะได้ว่า

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

ดังนั้นการแวะผ่านต้นไม้แบบหลังลำดับในภาพข้างต้น จากนิยามข้างบนนี้แสดงได้ผังข้างล่างนี้ ผลที่ได้ก็คือ ลำดับ E, C, H, F, G, D, B, J, L, K, I, A



จากนิยามการแวะผ่านแบบหลังลำดับข้างบนนี้สามารถเขียนเป็นโปรแกรมแบบเวียนเกิดได้ดังนี้

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

อ้างอิง

1. สมชาย ประสิทธิ์จูตระกูล, *การออกแบบและการวิเคราะห์อัลกอริทึม*. สำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ, 2549.
2. โครงการตำราวิทยาศาสตร์และคณิตศาสตร์มูลนิธิ สอวน., *โครงสร้างข้อมูลและอัลกอริทึม*. มูลนิธิ สอวน., 2548.
3. ชิดชนก เหลือสินทรัพย์. *Analysis & Design of Algorithms*. โรงพิมพ์ ดี แอล เอส กรุงเทพ ฯ, 2543
4. Cormen, T. H., C. E. Leiserson and R. L. Rivest, *Introduction to Algorithm*. MIT Press, 1990.
5. Anany Levitin, *Introduction to The Design & Analysis of algorithms*. Pearson Education, Inc., 2007.