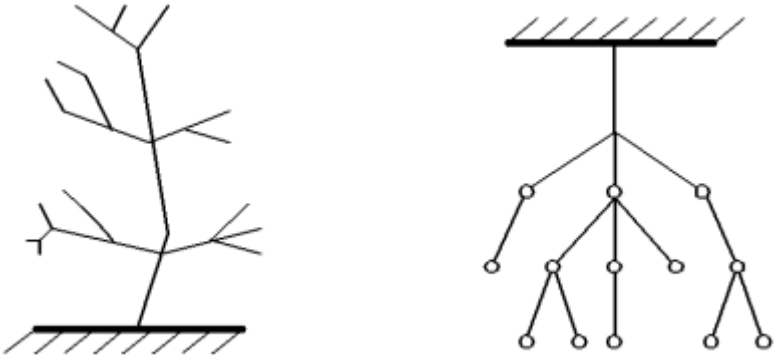
3.2 โครงสร้างต้นไม้ (Tree)

โครงสร้างต้นไม้ (Tree) เป็นโครงสร้างชนิดไม่เชิงเส้นที่สำคัญอย่างหนึ่งของโครงสร้างข้อมูล มีลักษณะคล้ายกิ่งก้านของต้นไม้ ต้นไม้ตามธรรมชาติจะงอกจากล่างไปบนส่วนโครงสร้างข้อมูลที่มีลักษณะต้นไม้นั้น จะวาดหรือให้เจริญจากบนลงมาล่าง จุดที่มีการแตกกิ่งก้านสาขาออกไปจะเรียกว่า โหนด (Node) โดยข่าวสารจะเก็บอยู่ที่โหนด กิ่งที่ต่อระหว่างโหนดจะแสดงความสัมพันธ์ระหว่างโหนดเรียกว่าลิงค์ (LInk) ซึ่งจุดปลายของกิ่งตามรูป 3.30 จะเรียกว่าโหนดด้วย



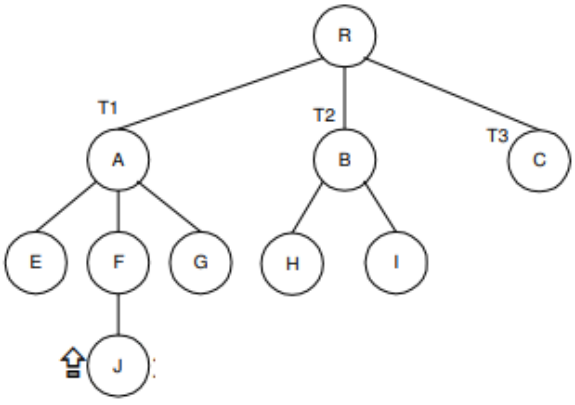
(ก) ลักษณะต้นไม้ทั่วไป (ข) ลักษณะต้นไม้ที่ใช้เก็บข้อมูล

รูปที่ 3.32 แสดงลักษณะโครงสร้างต้นไม้

นิยามของโครงสร้างต้นไม้

- มีโหนดพิเศษโหนดหนึ่งเรียกว่า รากหรือรูท (Root Node), R

- โหนดอื่น ๆ ที่ไม่ใช่รูท สามารถถูกแบ่งย่อยออกเป็น N กลุ่ม โดยที่แต่ละกลุ่มไม่มีโหนดร่วมกันเลย สมมติให้ชื่อแต่ละกลุ่มเป็น T_1 , T_2 , ... T_N ($N \geq 1$) ซึ่งแต่ละกลุ่มก็เป็นต้นไม้เหมือนกัน แต่จะเรียกว่าเป็น ต้นไม้ย่อย (Sub Tree) ของโหนด R ดังรูป 3.31



3.33 ลักษณะโครงสร้างของต้นไม้

จะเห็นได้ว่า

R เป็นรูทโหนดของต้นไม้ย่อย A,B,C

A เป็นรูทโหนดของต้นไม้ย่อย E,F,G

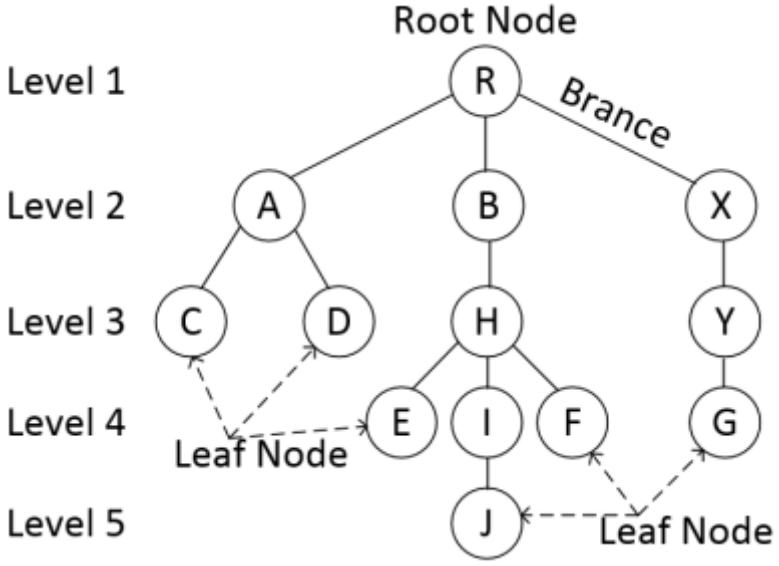
F เป็นรูทโหนดของต้นไม้ย่อย J

B เป็นรูทโหนดของต้นไม้ย่อย H และ I

3.2.1 องค์ประกอบของต้นไม้

การเรียกชื่อส่วนต่าง ๆ ของต้นไม้

ลักษณะของโครงสร้างต้นไม้มีลักษณะเหมือนการลำดับบรรพบุรุษ ดังนั้นชื่อที่ใช้จึงได้มาจากการลำดับบรรพบุรุษ เช่น พ่อ ลูก หลาน (Father , Son, Grandson) ที่ใช้บ่อยจะเป็นความสัมพันธ์ของพ่อ-ลูก (หรือ Father - Son)



รูปที่ 3.34 การเรียกชื่อส่วนต่าง ๆ ของโครงสร้างต้นไม้

ระดับของโหนด (Level)

ระดับของโหนดหนึ่ง ๆ แสดงถึงหน่วยระยะทางตามแนวดิ่งของโหนดนั้นว่าอยู่ห่างจากรูทโหนดเท่าไร ถ้ากำหนดว่ารูทโหนดของต้นไม้นั้นอยู่ที่ระดับ 1 และกิ่งทุกกิ่งมีความยาวเท่ากันหมดคือยาว 1 หน่วย เลขระดับของโหนดใด ๆ คือจำนวนกิ่งที่น้อยที่สุดจากรูทโหนดบวกหนึ่ง เช่น F มีเลขระดับเป็น 4 เป็นต้น

ดีกรีของโหนด (Degree)

ดีกรีของโหนด คือจำนวนต้นไม้ย่อยของโหนดนั้น จากรูป โหนด X มีดีกรี 1 โหนด A มีดีกรี 2 ส่วนโหนด H มีดีกรี 3 โหนด B มีดีกรี 1 และ โหนด E มีดีกรี 0 เป็นต้น

โหนดที่เป็นใบ (Leaf Node หรือ TermInal Node)

โหนดที่เป็นใบ หมายถึง โหนดที่มีดีกรีเป็น 0 หรือโหนดที่ไม่มีลูก เช่นโหนด C,D,E,J,F และ G ส่วนโหนดที่มีดีกรีไม่เท่ากับ 0 เรียกว่า โหนดภายในหรือ Interior Node หรือ Branch Node

กิ่งหรือเส้นเชื่อม (Branch หรือ Edge)

ได้แก่เส้นเชื่อมระหว่างโหนด พ่อ-ลูก

Predecessor และ Successor (หรือ Ancestor และ Descendant)

Predecessor หมายถึง ผู้มาก่อน หรือโหนดบน

Successor หมายถึง ผู้มาทีหลัง หรือโหนดล่าง

ตามรูป 3.32 R, B, H จะเป็น Predecessor ของโหนด E, I, F, J โหนด E และ F ไม่มี Successor ส่วนโหนด J เป็น Successor ของโหนด I และสรุปได้ว่าโหนด I จะเป็นPredecessor ของโหนด (หรือ J เป็น Successor ของโหนด) ถ้าโหนด I และโหนด J มีกิ่ง (หนึ่งกิ่งหรือมากกว่า) เชื่อมถึงกันและโหนด I มีเลขหมายระดับน้อยกว่าโหนด J

Immediate Successor หรือ Son ของโหนด i

Immediate Successor คือโหนดทั้งหลายของต้นไม้ย่อย i ที่มีค่าระดับต่ำกว่าโหนด i อยู่หนึ่ง เช่น Son ของโหนด H คือโหนด E, I และ F

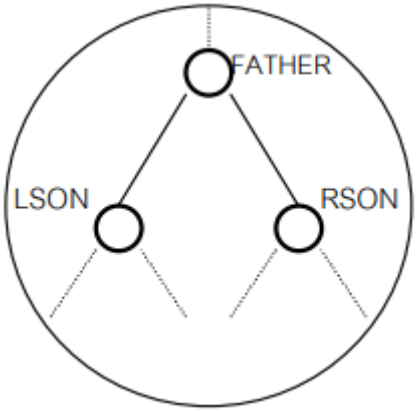
Immediate Predecessor หรือ FARTHER โหนด i

Immediate Predecessor คือโหนดที่มีค่าระดับสูงกว่าโหนด i อยู่หนึ่ง เช่น FARTHER ของโหนด J คือ โหนด i , Father ของโหนด i คือโหนด H เป็นต้น

3.2.2 ต้นไม้ไบนารี

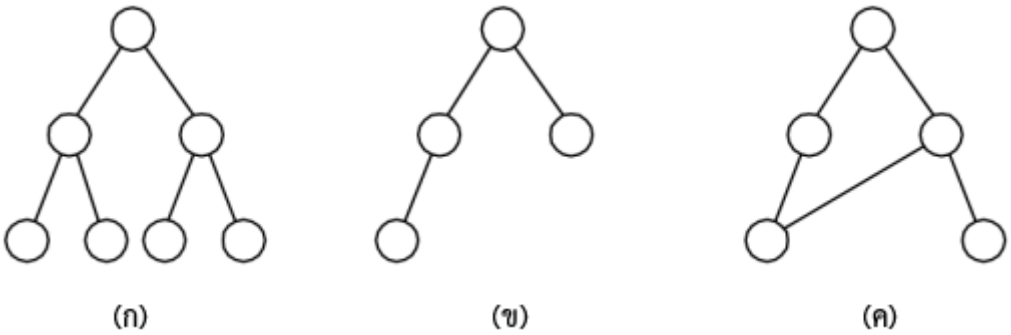
ต้นไม้ทั่วไปมีลักษณะเป็นโครงสร้างแบบต้นไม้ที่มีดีกรีหรือโหนดลูกมากกว่า 2 ส่วน ต้นไม้แบบไบนารีเป็นต้นไม้ที่มีโหนดที่หนึ่งเป็นรากเพียงโหนดเดียว โดยมีลูกไม่เกิน 2 โดยแยกเป็นต้นไม้ย่อยทางซ้าย (Left subtree) และต้นไม้ย่อยทางขวา (Right subtree) โดยต้นไม้ย่อยทั้งสองจะไม่มีโหนดพ่อร่วมกัน

โครงสร้างต้นไม้ที่ใช้ในการจัดการข่าวสารโดยคอมพิวเตอร์นั้น ส่วนใหญ่จะเป็นแบบโครงสร้างต้นไม้ไบนารี โดยจากโหนดที่หนึ่งจะมีความสัมพันธ์เป็นพ่อ (FARTHER) และมีอีก 2 โหนดต่อลงไปเป็นลูกโหนดทางซ้าย (Left son หรือ LSon) และโหนดทางขวา (Right Son หรือ RSon) ดังรูป 3.33



รูปที่ 3.35 โครงสร้างต้นไม้ไบนารี

ตัวอย่างของต้นไม้ไบนารีดูได้จากรูปข้างล่าง โดยรูป (ก) , (ข) นั้นเป็นต้นไม้ไบนารี ส่วนรูป (ค) นั้นไม่ใช่ต้นไม้ไบนารีเนื่องจากมีวงจรปิด



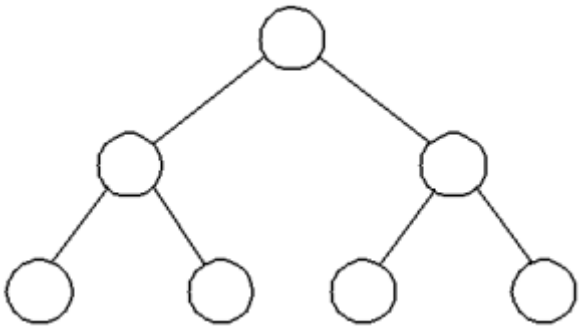
รูปที่ 3.36 แสดงการพิจารณาลักษณะต้นไม้ไบนารี

3.2.2.1 ต้นไม้ไบนารีสมบูรณ์ (Complete BInary Tree)

ต้นไม้ไบนารีสมบูรณ์ หมายถึงต้นไม้ไบนารีที่แต่ละโหนดภายในมีโหนดย่อยซ้าย และขวา (นั่นคือ แต่ละโหนดภายในมี Left son , Right son และ โหนดใบ left nodes) ทั้งหลายอยู่ในระดับที่ N ดังรูป 3.35 ซึ่งเป็นต้นไว้ไบนารีสมบูรณ์ที่มี 3 ระดับ ซึ่งความสัมพันธ์ระหว่างระดับ (L) กับจำนวนโหนด (N) ในต้นไม้ไบนารีแบบสมบูรณ์ จะเขียนได้ดังนี้

$$N = 2^{L} - 1$$

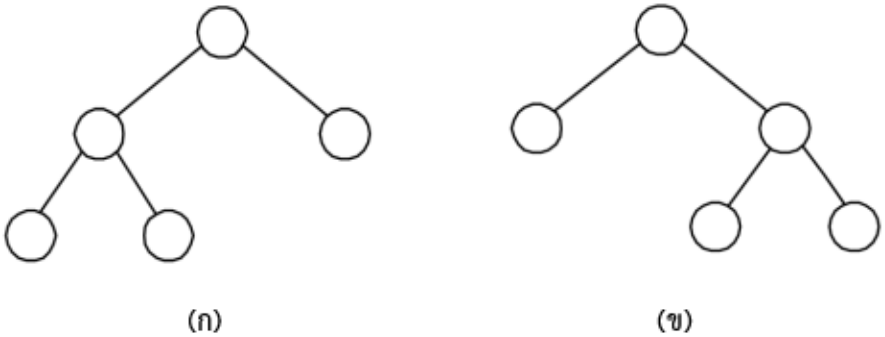
ดังนั้นต้นไม้ดังรูป 3.35 จึงมีจำนวนโหนด $N = 2^3 - 1 = 7$ โหนด



รูปที่ 3.37 แสดงต้นไม้ไบนารีสมบูรณ์

3.2.2.2 ต้นไม้ที่มีแบบแผนและไม่มีแบบแผน

ต้นไม้ (ก) และ (ข) จะเป็นต้นไม้ที่มีแบบแผนถ้าโหนดต่าง ๆ ของต้นไม้นั้นมีความสัมพันธ์ที่แน่นอน เช่น ค่าน้อยอยู่ทางซ้าย ค่ามากอยู่ทางขวา เช่น ต้นไม้ไบนารีเพื่อการค้นหา (BInary Search Tree) ทำให้ต้นไม้ (ก) และ (ข) มีค่าไม่เท่ากัน แต่หากต้นไม้ (ก) และ (ข) ไม่สนใจในความสัมพันธ์เชิงตำแหน่งของโหนด ก็จะเป็นต้นไม้ที่ไม่มีแบบแผน ต้นไม้ (ก)และ (ข) จะมีค่าเท่ากันเพราะมีจำนวนโหนดเท่ากัน เป็นต้น

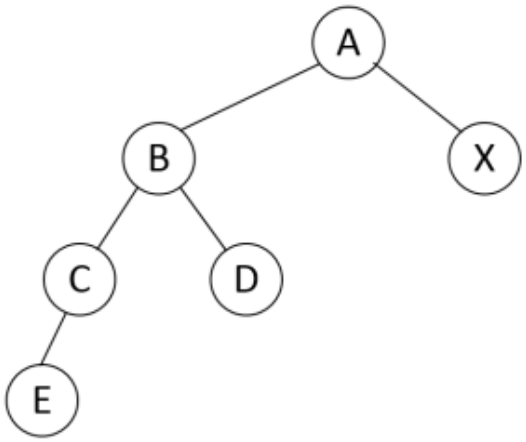


รูปที่ 3.38 แสดงความเป็นแบบแผนของต้นไม้

3.2.3 การแทนโครงสร้างต้นไม้ในคอมพิวเตอร์

3.2.3.1 การแทนต้นไม้โดยอาเรย์

การแทนต้นไม้โดยอาเรย์ จะจำลองการแทนด้วยอาเรย์ 3 อาเรย์ เพื่อใช้เก็บข่าวสาร (INFO) ลูกทางซ้าย (LSon) และลูกทางขวา (RSon) โดยแต่ละอาเรย์จะมีจำนวนช่องตามจำนวนโหนด (N) ค่าตัวเลขที่เติมไว้ได้มาจากการนำข้อมูล INFO ของต้นไม้ และเลขลำดับของช่อง (Subscript) ของอาเรย์ที่ LSon, RSon นั้นเก็บอยู่มาใส่ และจะใช้ ^ แทนส่วนลิงค์ที่เป็นศูนย์ หรือใช้สัญลักษณ์พิเศษอื่น ๆ ในที่นี้จะแทนด้วย ขีด (-) ดังรูป



รูปที่ 3.39 โครงสร้างต้นไม้

	1	2	3	4	5	6
INFO	A	B	X	C	D	E
LSON	2	4	-	6	-	-
RSON	3	5	-	-	-	-

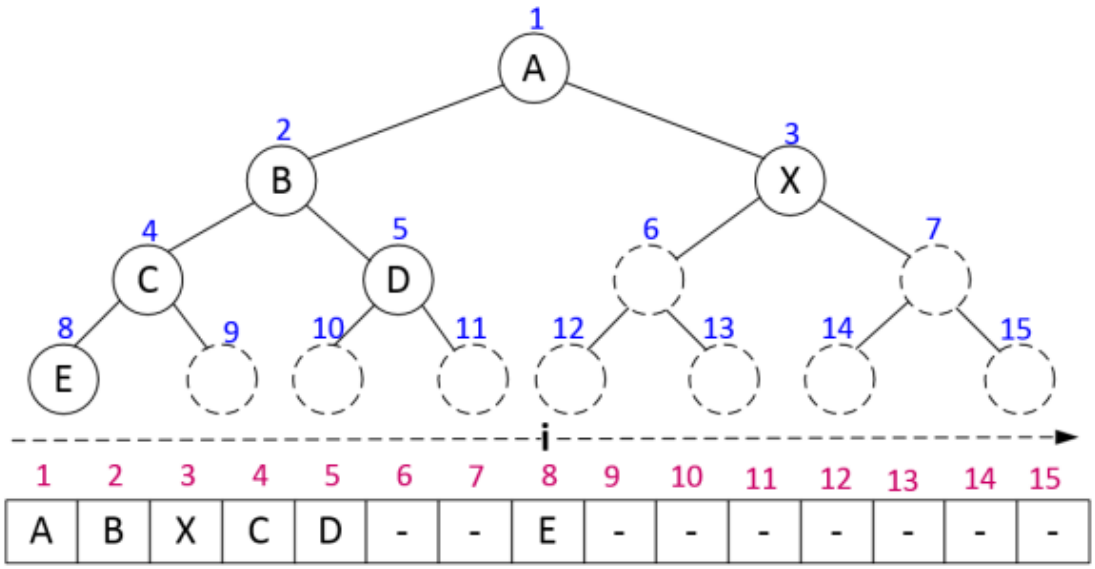
รูปที่ 3.40 แสดงการแทนโดยใช้อาเรย์

3.2.3.2 การแทนต้นไม้แบบเรียงโหนด

จากการแทนโดยอาเรย์ข้างต้น จะเห็นว่ามีช่อง LSon และ RSon ที่เป็นค่าว่างอยู่จำนวนหนึ่ง ซึ่งหากต้นไม้นั้นมีจำนวนโหนดมากก็จะทำให้ต้องเก็บค่าว่างเหล่านี้มากขึ้นด้วยทำให้เปลืองพื้นที่ ดังนั้นจึงมีวิธีการเก็บโดยใช้อาเรย์ซึ่งจะไม่เก็บตัวชี้ LSon และ RSon แต่จะใช้การคำนวณหาตำแหน่งความสัมพันธ์ของต้นไม้ เช่น Father, LSon และ RSon การแทนแบบนี้เรียกว่า **“การแทนแบบเรียงโหนด”** ซึ่งการแทนแบบนี้จะใช้ **อาเรย์ขนาด 1 มิติ** และเหมาะกับต้นไม้ที่เป็นต้นไม้ที่มีความสมบูรณ์

(Full BInary) ซึ่งจะมีจำนวนโหนดสูงสุด = N โดยหาจาก $N=2^L - 1$ แต่หากไม่ใช่ต้องทำการต่อเติมโหนดเปล่า ๆ ให้เป็นต้นไม่สมบูรณ์

การแทนจะเริ่มต้นด้วยการให้หมายเลขแอดเดรสแก่แต่ละโหนด ตั้งแต่ 1 , 2 , 3 ... ไปเรื่อยๆ จนถึง k การให้ตัวเลขในแต่ละระดับจะให้จากซ้ายมาขวา โดยให้รูทโหนดมีหมายเลข 1 เสมอ การให้ตัวเลขจะต้องถือว่าต้นไม้ใบนารีเป็นต้นไม้ไบนารีแบบสมบูรณ์ดังนั้นรูปต้นไม้หากไม่เป็นต้นไม้สมบูรณ์จะต้องถูกต่อเติมให้เป็นต้นไม้ไบนารีแบบสมบูรณ์ จึงจะให้ตัวเลขที่อยู่แก่โหนดได้



รูปที่ 3.41 แสดงการแทนแบบเรียงโหนด

โดยทั่วไปแล้วจำนวนช่องอาเรย์มีใช้จะมีค่าเท่ากับจำนวนโหนด $N = 2^L - 1$ ช่อง และเมื่อแทนโดยอาเรย์แล้ว จะต้องมีวิธีคำนวณหาความสัมพันธ์ "พ่อ-ลูก" ของโหนด และเมื่อกำหนดให้ i คือเลขแอสเดรสตำแหน่ง ดังนั้น FATHER, LSON และ RSON ของโหนดใดๆ จะหาได้โดยความสัมพันธ์ของสมการ และอสมการ(สมการข้อจำกัด) ดังต่อไปนี้

- Father(i) = |i/2| ; อสมการ = $1 < i \leq N$
- LSon(i) = 2i ; อสมการ = $1 \leq i \leq (2^{(L-1)}-1)$
- RSon(i) = (2i +1) ; อสมการ = $1 \leq i \leq (2^{(L-1)}-1)$

3.2.3.3 การแทนต้นไม้โดยพอยน์เตอร์

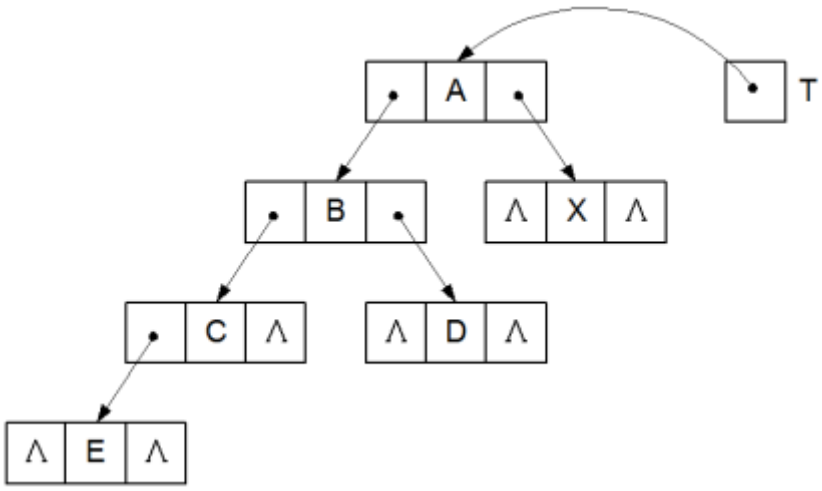
ทำได้โดยให้แต่ละโหนดมีโครงสร้างดังรูป 3.42

รูปที่ 3.42 โครงสร้างโหนดสำหรับต้นไม้ไบนารี

LLINK หรือ RSon เป็นพอยน์เตอร์ชี้ไปยังต้นไม้ย่อยทางซ้าย ส่วน RLINK หรือRSon เป็นพอยน์ตอร์ชี้ไปยังต้นไม้ย่อยทางขวา และต้นไม้ไบนารีทั้งต้นจะแทนด้วย T โดยที่ T เป็นพอยน์เตอร์ชี้ไปยังรูทโหนด

ถ้า T = ^ แสดงว่าต้นไม้ไบนารีนั้นว่างเปล่า

T ≠ ^ แสดงว่าต้นไม้ไบนารีนั้นไม่ว่างเปล่า และ T จะเป็นแอดเดรสของรูทโหนด (^ เป็นค่าที่โหนดนั้นไม่มีลูก) จะได้ลักษณะการแทนดังรูป 3.41



รูปที่ 3.43 แสดงการแทนต้นไม้โดยพอยน์เตอร์

3.2.3.4 การท่องเข้าไปในโครงสร้างต้นไม้ไบนารี (Tree Traversal)

การนำต้นไม้ไบนารีมาใช้ประโยชน์นั้น หมายความว่า ต้องมีวิธี เดิน หรือ ท่อง(Traverse) เข้าไปในต้นไม้อย่างมีแบบแผน เพื่อไป เยี่ยม โหนดต่าง ๆ โหนดละ 1 ครั้ง คำว่า “เยี่ยม” หมายถึงว่าจะไปยังโหนดเพื่อประมวลผลบางอย่างที่ต้องการกระทำกับโหนดนั้นเช่น หาข่าวสาร ดังนั้นผลลัพธ์จากการเดินเข้าไปในต้นไม้คือจะได้ข่าวสารที่เก็บ (หรือชี้)โดยโหนดเหล่านั้นออกมาในรูปเชิงเส้น

ความคิดเริ่มต้นของการเดินก็คือ ต้องเยี่ยมแต่ละโหนดพร้อมทั้งต้นไม้ย่อยทางซ้ายและทางขวาของโหนดนั้น ตามแนวความคิดนี้ จะแยกต้นไม้ไบนารีออกเป็น 3 ส่วน คือ R, TL , และ TR โดยที่ R จะแทนรูทโหนด TL จะแทนต้นไม้ย่อยทางซ้าย และ TR จะแทนต้นไม้ย่อยทางขวา จะเห็นได้ว่ามีวิธีเดินอยู่ 6 วิธีคือ

1. Pre Order: R-TL-TR
2. In Order : TL-R-TR
3. Post : TL-TR-R
4. Reverse Pre Order : R-TR-TL
5. Reverse In Order : TR-R-TL
6. Reverse Post Order : TR-TL-R

จะมีวิธีเดิน 6 วิธี แต่ จะสนใจเพียง 3 วิธีแรกเท่านั้น ซึ่ง 3 วิธีแรกมีชื่อเรียกและลักษณะการเดินดังนี้

(1) เดินแบบพรีออร์เดอร์ (Pre-Order Traversal) R-TL-TR

- เยี่ยม R
- เดินเข้าไปใน TL (ของ R) อย่างพรีออร์เดอร์
- เดินเข้าไปใน TR (ของ R) อย่างพรีออร์เดอร์

(2) เดินแบบอินออร์เดอร์ (In-Order Traversal) TL-R-TR

- เดินเข้าไปใน TL (ของ R) อย่างอินออร์เดอร์
- เยี่ยม R
- เดินเข้าไปใน TR (ของ R) อย่างอินออร์เดอร์

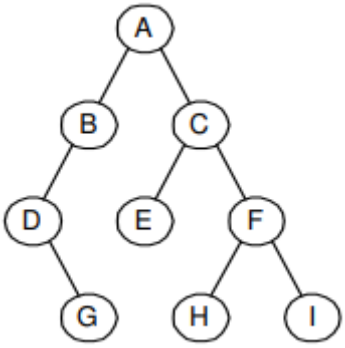
(3) เดินแบบโพสต์ออร์เดอร์ (Post-Order Traversal) TL-TR-R

- เดินเข้าไปใน TL (ของ R) อย่างโพสต์ออร์เดอร์
- เดินเข้าไปใน TR (ของ R) อย่างโพสต์ออร์เดอร์
- เยี่ยม R

ส่วนอีก 3 วิธี เกิดจากการสับเปลี่ยนตำแหน่ง TL และ TR ของ 3 วิธีที่กล่าวมาแล้วดังนั้น 3 วิธีหลังจะมีชื่อคล้าย ๆ กับ 3 วิธีแรก เพียงแต่มีคำว่า Reverse เพิ่มเข้าไปเท่านั้น

อัลกอริทึมการท่องแบบพรีออเดอร์ (Pre-Order Traversal : R-TL-TR)

สมมติมีต้นไม้ไบนารีตามรูป 3.44



รูป 3.44 ต้นไม้ไบนารี

วิธีการเริ่มที่รูทโหนด A แล้วให้เยี่ยมโหนด A ก่อน คำว่า เยี่ยม จะหมายถึง พิมพ์ชื่อโหนดออกมา ต่อไปให้เดินเข้าไปใน TL ของโหนด A ไปอยู่ที่ B (เนื่องจาก B เป็นรูทโหนดของต้นไม้ย่อยทางซ้ายของ A) ซึ่งเท่ากับ ตั้งต้น ทำ R-TL-TR ใหม่ ดังนั้นโหนด B จะตรงกับตำแหน่ง R ใน R-TL-TR นั่นคือ ต้องเยี่ยมโหนด B เมื่อเยี่ยมโหนด B แล้ว จะเดินเข้าไปใน TL ของโหนด B ไปอยู่ที่ D แต่เนื่องจาก TL ของโหนด D ว่างเปล่า ดังนั้นจึงทำตำแหน่ง TR ใน R-TL-TR ต่อไป TR ของโหนด D คือ G จึงได้ค่า G พิมพ์ออกมา สรุปได้ว่าในขณะนี้ชุดของโหนดที่พิมพ์ออกมา คือ ABDG

ต่อไปให้เดินเข้าไปใน TL ของ G ที่ว่างเปล่า ดังนั้นจึงข้ามไปเดิน TR ของ G ซึ่งก็ว่างเปล่าอีกเช่นกัน ณ จุดนี้เท่ากับ ได้เดิน TL ของโหนด A ครบแล้ว ดังนั้นจึงข้ามไปทำ TR ของโหนด A นั่นคือไปตั้งต้นที่โหนด C แล้วเริ่มทำการเดินตามนิยาม R-TL-TR ใหม่อีก โดยเริ่มเยี่ยมโหนด C แล้วเดินเข้าไปใน TL ของโหนด C ต่อไปเริ่มเยี่ยมโหนด E จากนั้นก็เดินเข้าไปใน TL ของโหนด E ซึ่งว่างเปล่า ต่อไปก็เดินเข้าไปใน TR ของโหนด E ซึ่งก็ว่างเปล่าอีกดังนั้นเท่ากับได้เดิน TL ของโหนด C ครบแล้วต่อไปก็เดินเข้าไปใน TR ของโหนด C อีกจนครบ ในที่สุดก็เท่ากับเดินเข้าไปใน TR ของโหนด A ครบแล้ว

เมื่อ เดินท่องต้นไม้ไบนารีอย่างพรีออร์เดอร์ (R-TL-TR) แล้วชุดข่าวสารที่พิมพ์ออกมาก็จะเป็น **ABDGCEFHI**

อัลกอริทึมการท่องแบบอินออร์เดอร์ (In-Order Traversal : TL-R-TR)

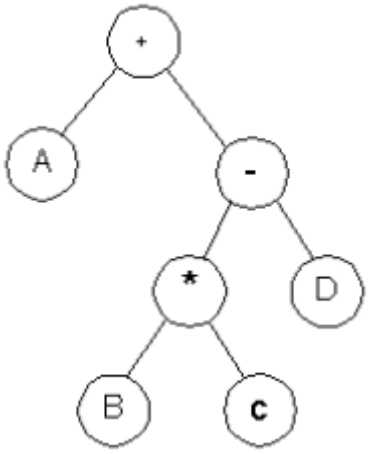
การเดินแบบนี้มีลักษณะคล้ายกับการเดินแบบพรีออร์เดอร์ นั่นคือต้องท่องต้นไม้ย่อยทางซ้าย (TL) ให้ครบก่อน แล้วจึงค่อยท่องต้นไม้ย่อยทางขวา (TR ได้ การเดินแบบอินออร์เดอร์นี้ จะเยี่ยมโหนด

ภายหลังจากที่ได้เดิน TL ครบหมดแล้วเท่านั้น ต้นไม้ไบนารีตามรูป 3.44 เมื่อเดินอย่างอินออร์เดอร์แล้วจะได้ชุดลำดับของโหนดที่เยี่ยมเป็น **DGBAECHFI**

อัลกอริทึมการท่องแบบโพสต์ออร์เดอร์ (Post-Order Traversal : TL-TR-R)

ตามรูป 3.44 เริ่มเดินเข้าไปใน TL จากโหนด A ถึงแค่โหนด D เพราะ Lson โหนดD ไม่มี จากนั้นก็เริ่มเดินเข้าไปใน TR ของโหนด D ซึ่งก็คือ G เป็นอันว่า ได้เดิน TL, TR ของโหนด D ครบ ต่อไปคือ R ซึ่งได้แก่การเยี่ยมโหนด D และได้ GD พิมพ์ออกมา เมื่อถึงตอนนี้ก็เท่ากับว่า ได้เดิน TL ของโหนด B ครบ จึงเริ่มเดินเข้าไปใน TR ของโหนด B แต่ไม่มีจึงพิมพ์ GDB ออกมา ณ จุดนี้เท่ากับว่า ได้เดิน TL ของโหนด A ครบแล้วต่อไปจึงเดิน TRของโหนด A ซึ่งจะได้โหนด (โดยเยี่ยม TL ของ C คือ : พิมพ์ออกมาตามลำดับเท่ากับว่าได้เดินเข้าไปใน TL ของ C ครบแล้ว จึงเดินไปยัง TR ของโหนด C คือ F และเยี่ยม TL ของF คือ H และเยี่ยม TR ของ F คือ ! ย้อนขึ้น R ซึ่งก็คือ F , C และโหนด A ตามลำดับเป็นอันว่าสิ้นสุดการเดินอย่างโพสต์ออร์เดอร์เข้าไปในต้นไม้นี้ จากรูป เมื่อเดินอย่างโพสต์ออร์เดอร์แล้วจะได้โหนดที่เยี่ยมออกมาเป็น **GDBEHIFCA**

ตัวอย่าง จงท่องต้นไม้ทั้ง 3 วิธี (Pre-Order, In-Order, Post-Order)



รูปที่ 3.45 ต้นไม้ไบนารี

Sol. ก. Pre-Order (R-TL-TR) = +A-*BCD

ข. In-Order (TL-R-TR) = A+B*C-D

ค. Post-Order (TL-TR-R) = ABC*D-+

อัลกอริทึมในการท่องเข้าไปในโครงสร้างต้นไม้ไบนารี

```
;ให้ pt เป็น poiter ชี้ Address โดยเริ่มต้นให้ชี้ที่ Root
Procedure Preorder(TreeNode pt)      ; R-TL-TR
        If (pt!=null) Then
                Visit(pt);                  ;เยี่ยม ROOT
                Preorder(pt=pt.left);       ;เยี่ยม LSON
                Preorder(pt=pt.right);      ;เยี่ยม RSON
        Endlf
End
Procedure Inorder(TreeNode pt)       ;การเดินเข้าแบบอินออร์เดอร์ (TL-R-TR)
        If (pt!=null) Then
                Inorder(pt=pt.left);        ; เยี่ยม LSON
                Visit(pt);                  ; เยี่ยม ROOT
                Inorder(pt=pt.right);       ; เยี่ยม RSON
        Endif
End
Procedure Postorder(TreeNode pt)     ; การเดินเข้าแบบโพสต์ออร์เดอร์ (TL-TR-R)
        if (pt!=null) begln
                Postorder(pt=pt.left);          ; เยี่ยม LSON
                Postorder(pt=pt.right);         ; เยี่ยม RSON
                Visit(pt);                      ;เยี่ยม ROOT
        Endif
End
```

การวางโครงสร้างโปรแกรมเก็บโครงสร้างต้นไม้แบบเรียงโหนด

ด้วยเหตุที่การแทนแบบเรียงโหนดจะเก็บโครงสร้างต้นไม้ไว้ในอาเรย์ 1 มิติ แล้วเก็บโหนดเรียงกันไปในอาเรย์ตั้งแต่โหนด 1,2,3.... ไปเรื่อย ๆ ดังนั้นจึงใช้คำสั่ง for ในการวนลูปเพื่อเก็บข้อมูลของแต่ละโหนดลงในอาเรย์ตามลำดับ

ส่วนการท่องในแต่ละแบบก็เขียนเป็นแบบเรียกตนเอง (Recursive) ตามตัวอย่างข้างต้น โดยเริ่มต้นท่องที่รูทโหนด (โหนดที่ 1) เป็นต้นไป และการหาค่าของ LSon และ RSon ก็ใช้สมการ LSon(i)=2i และ RSon(i)=2i+1

3.2.3.5 โปรแกรมการแทนไม้แบบเรียงโหนดและการท่อง

```
/* Program create Tree structure by "NODE SEQUENCE Method".
Can shows the result when traversaled by..
 1. Pre Order
 2. In Order
 3. Post Order
 4. Can use for 5 Level maximum of tree (N=31)
*/
#include <stdio.h> //use printf
#include <conio.h> //use getch
#include <stdlib.h> //use random
#include <math.h> //use pow
#define MaxNode 100 // Define Max Node of Tree
int N,data[MaxNode]; // Declare Array for keep data of Tree
char ch;

void CreateTreeNS(int n)
 {
    int i,temp;
    for (i=1;i<=n;i++)
    {
        temp=1+rand() % 99; //random difference number 1..99
        data[i]=temp;
    }
 }

void ShowArray()
 {
    int i=1;
    while (data[i] != NULL)
    {
        printf("[%i]%d ",i,data[i]); i++;
    }
    printf("\n============================================================
====================\n");
 }

void ShowTree()
 {
```

```
    int j,level,start,ends;
    j=1;
    level=1; //Start al Level 1
    printf("\n");
    while (data[j] != NULL)
    {
        start=pow(2,level)/2; //Calculate START Node of this level
        ends=pow(2,level)-1; //Calculate END Node of this level
        for (j=start;j<=ends;j++)
        {

            if(data[j] != NULL)
            {
                switch (level)
                {
                    case 1 : printf("%40d",data[j]);
                                break;
                    case 2 : if (j==2)
                                printf("%20d",data[j]);
                             else
                                printf("%40d",data[j]);
                             break;
                    case 3 : if(j==4)
                                printf("%10d",data[j]);
                             else
                                printf("%20d",data[j]);
                             break;
                    case 4 : if(j==8)
                                printf("%5d",data[j]);
                             else
                                printf("%10d",data[j]);
                             break;
                    case 5 : if(j==16)
                                printf("%d",data[j]);
                             else
                                printf("%5d",data[j]);
                             break;
                }
            }
        }
        printf("\n\n"); //Line feed
        level++;
    }
 }

void PreOrder(int i)
```

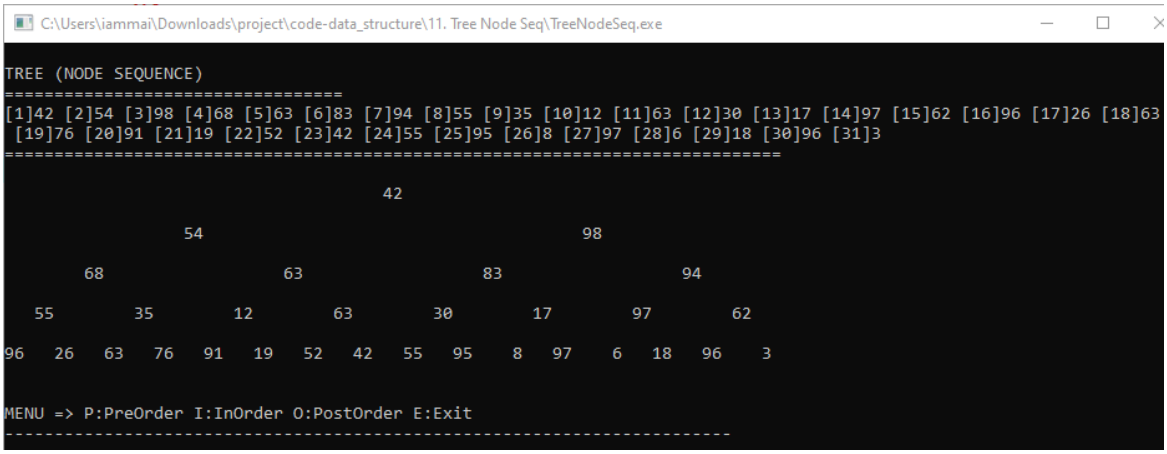
```
 {
    int info,lson,rson;
    info=data[i]; // Root info
    if (info != NULL) //if INFO NOT NULL
    {
        printf(" %d",data[i]); //Display INFO
        lson=2*i; //Calculate LSON
        rson=2*i+1; //Calculate RSON
        PreOrder(lson); //Call left Son by PreOrder
        PreOrder(rson); //Call Right Son by PreOrder
    }
 }

void InOrder(int i)
 {
    int info,lson,rson;
    info=data[i]; // Root info
    if (info != NULL) //if INFO NOT NULL
    {
        lson=2*i; //Calculate LSON
        rson=2*i+1; //Calculate RSON
        InOrder(lson); //Call left Son by InOrder
        printf(" %d",data[i]); //Display INFO
        InOrder(rson); //Call Right Son by InOrder
    }
 }

void PostOrder(int i)
 {
    int info,lson,rson;
    info=data[i]; // Root info
    if (info != NULL) //if INFO NOT NULL
    {
        lson=2*i; //Calculate LSON
        rson=2*i+1; //Calculate RSON
        PostOrder(lson); //Call left Son by PostOrder
        PostOrder(rson); //Call Right Son by PostOrder
        printf(" %d",data[i]); //Display INFO
    }
 }

int main()
{
    N=31;
    CreateTreeNS(N); //Create N Node
    while (ch != 'E')
```

```
    {
        printf("\nTREE (NODE SEQUENCE)\n");
        printf("==================================\n");
        ShowArray();
        ShowTree();
        printf("\nMENU => P:PreOrder I:InOrder O:PostOrder E:Exit");
        printf("\n-------------------------------------------------------
--------------------\n");
        ch=getch();
        switch (ch)
        {
        case 'P' :  ShowTree();
                    printf("PRE ORDER TRAVERSAL : ");
                    PreOrder(1);
                    printf("\n"); //Line feed
                    break;
        case 'I' :  ShowTree();
                    printf("IN ORDER TRAVERSAL : ");
                    InOrder(1);
                    printf("\n"); //Line feed
                    break;
        case 'O' :  ShowTree();
                    printf("POST ORDER TRAVERSAL : ");
                    PostOrder(1);
                    printf("\n"); //Line feed
                    break;
        } //End Switch...case
    } //End While
return(0);
}//End MAIN
```



รูปที่ 3.46 แสดงผลการทำงานของโปรแกรมการแทนต้นไม้แบบเรียงโหนด

การวางโครงสร้างโปรแกรมเก็บโครงสร้างต้นไม้แบบพอยน์เตอร์

ด้วยเหตุที่ต้องการสร้างต้นไม้แบบ FULL คือมีจำนวนโหนดเต็ม และเรียงกันเหมือนกับแบบเรียงโหนด ดังนั้นจึงใช้เทคนิคของการหา Father(i) แบบเรียงโหนด โดยทำการเก็บค่า Address ของโหนดที่เข้ามาตามลำดับในอาเรย์ในตำแหน่ง 1 2 3 ไปเรื่อย ๆเพื่อให้ง่ายต่อการหาพ่อ แล้วจึงเปลี่ยนพอยน์เตอร์ของ SON ของ Father เพื่อชี้โหนดใหม่ที่เพิ่มเข้าไป (New Node) โดยจะให้โหนดใหม่อยู่ทางซ้ายหรือขวาก็สามารถตรวจสอบค่าของพอยน์เตอร์ใน LSON และ RSON ก่อนที่จะกำหนดได้โดยง่าย

3.2.3.6 โปรแกรมการแทนไม้แบบพอยน์เตอร์และการท่อง

```
/* Program create Tree structure by "NODE POINTER Method".
Can shows the result when traversal by..
 1. Pre Order
 2. In Order
 3. Post Order
Note.- Use for 5 Level maximum of tree (N=31)
*/
#include <stdio.h> //use printf
#include <conio.h> //use getch
#include <stdlib.h> //use random, malloc
#include <math.h> //use pow
#define MaxNode 40

struct Node {//Declare structure of Tree node
    int info;
    struct Node *lson;
    struct Node *rson;
 };

struct Node *T, *address[MaxNode]; // Declare pointer T of Tree node
int i,N,info[MaxNode];
char ch;

Node *Allocate() {//Allocate 1 node from storage pool and return
pointer of node
    struct Node *temp;
    temp=(Node*)malloc(sizeof(Node)); //Allocate node by size declare
    return(temp);
 }

void CreateTreeNP(int n) {
    int i,temp,Father;
```

```
    struct Node *p, *FatherPT;
    T=NULL; //Set start of T Pointer
    for (i=1;i<=n;i++){
        p=Allocate(); // Allocate NODE p
        temp=1+rand() % 99; //random difference number 1..99
        info[i]=temp; //Keep data for Check Correcting of Sequence
        address[i]=p; //Keep Address of Node Sequence
        p->info=temp; //Keep data to INFO
        p->lson=NULL; //Set default LSON=NULL
        p->rson=NULL; //Set default RSON=NULL
        if (T==NULL) { //Check for T=NULL?
            T=p; //Set T point to first node for begining of Treer
        }
        else{
            Father=(i/2); //Calculate FATHER
            FatherPT=address[Father]; //Get pointer of Father Node
            if(FatherPT->lson == NULL)
                FatherPT->lson=p; //Link LSON to new node
            else
                FatherPT->rson=p; //link RSON to new node
        }
    }
 }

void ShowTree(){
    int j,level,start,ends;
    j=1;
    level=1; //Start al Level 1
    printf("\n");
    while (info[j] != NULL) {
    start=pow(2,level)/2; //Calculate START Node of this level
    ends=pow(2,level)-1; //Calculate END Node of this level
    for (j=start;j<=ends;j++)
        if(info[j] != NULL) {
            switch (level) {
                case 1 : printf("%40d",info[j]);
                        break;
                case 2 : if (j==2)
                            printf("%20d",info[j]);
                        else
                            printf("%40d",info[j]);
                        break;
                case 3 : if(j==4)
                            printf("%10d",info[j]);
                        else
                            printf("%20d",info[j]);
```

```
                        break;
                case 4 : if(j==8)
                            printf("%5d",info[j]);
                        else
                            printf("%10d",info[j]);
                        break;
                case 5 : if(j==16)
                            printf("%d",info[j]);
                        else
                            printf("%5d",info[j]);
                        break;
            }
        }
        printf("\n"); //Line feed
        level++;
    }
 }

void PreOrder(struct Node *i)
 {
    if (i != NULL) {//if i NOT NULL
        printf(" %d",i->info); //Display INFO
        PreOrder(i->lson); //Call left Son by PreOrder
        PreOrder(i->rson); //Call Right Son by PreOrder
    }
 }

void InOrder(struct Node *i)
 {
    if (i != NULL) { //if i NOT NULL
        InOrder(i->lson); //Call left Son by InOrder
        printf(" %d",i->info); //Display INFO
        InOrder(i->rson); //Call Right Son by InOrder
    }
 }

void PostOrder(struct Node *i)
 {
    if (i != NULL) {//if i NOT NULL
        PostOrder(i->lson); //Call left Son by PostOrder
        PostOrder(i->rson); //Call Right Son by PostOrder
        printf(" %d",i->info); //Display INFO
    }
 }
int main()
{
```

```
    N=31;
    CreateTreeNP(N);
    while (ch != 'E') //Loop until 'E' Pressed
    {
        printf("\nPROGRAM TREE(Node Pointer) \n");
        printf("================================= \n");
        printf("N : %d\n",N);
        printf("Sequence of data : ");
        for (i=1;i<=N;i++) //Show Data Sequence
            printf("%d ",info[i]);
        ShowTree(); //Show tree structure
        printf("\nMENU => P:PreOrder I:InOrder O:PostOrder E:Exit");
        printf("\n-------------------------------------------------------
--------------------\n");
        ch=getch();
        switch (ch)
        {
            case 'P' :  printf("PRE ORDER TRAVERSAL : ");
                        PreOrder(T);
                        printf("\n-------------------------------------
-----------------------------------\n");
                        break;
            case 'I' :  printf("IN ORDER TRAVERSAL : ");
                        InOrder(T);
                        printf("\n-------------------------------------
-----------------------------------\n");
                        break;
            case 'O' :  printf("POST ORDER TRAVERSAL : ");
                        PostOrder(T);
                        printf("\n-------------------------------------
-----------------------------------\n");
                        break;
        } //End Switch...case
    } //End While
return(0);
}//End MAIN
```

```
PROGRAM TREE(Node Pointer)
==================================
N : 31
Sequence of data : 42 54 98 68 63 83 94 55 35 12 63 30 17 97 62 96 26 63 76 91 19 52 42 55 95 8 97 6 18 96 3
                                  42
                54                                      98
        68                63                    83                94
   55        35        12        63        30        17        97        62
96   26   63   76   91   19   52   42   55   95    8   97    6   18   96    3

MENU => P:PreOrder I:InOrder O:PostOrder E:Exit
--------------------------------------------------------------------------
```

รูปที่ 3.47 แสดงผลการทำงานของโปรแกรมการแทนต้นไม้โดยพอยน์เตอร์