บทที่ 2

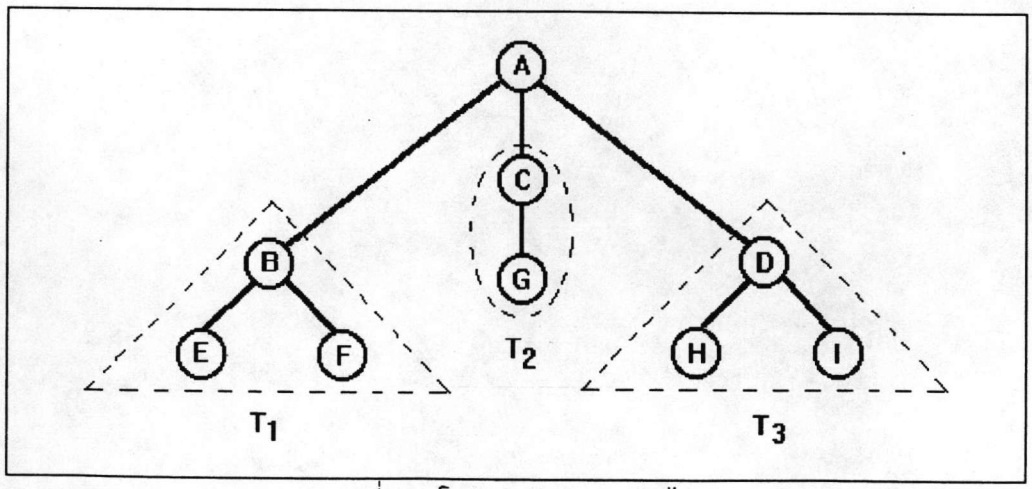
แนวคิดและทฤษฎีสำคัญ

โครงสร้างข้อมูล

"โครงสร้างข้อมูล" (DATA STRUCTURE) หมายถึง รูปแบบของการจัดระเบียบของข้อมูล ซึ่งอาจจะจัดทำได้หลายรูปแบบด้วยกัน ที่สำคัญ ๆ ได้แก่ โครงสร้างแบบแถวลำดับ แบบกองซ้อน แบบแถวคอย (Queue) แบบบัญชีรายการเชื่อมโยง (Link List) และโครงสร้างข้อมูลแบบต้นไม้ (ทักษิณา สวนานนท์, 2536)

โครงสร้างข้อมูลแบบต้นไม้

โครงสร้างข้อมูลแบบต้นไม้มีลักษณะที่สมชื่อของตัวเอง เพราะมีลักษณะคล้ายกิ่งก้านของต้นไม้ ต้นไม้ตามธรรมชาติจะงอกจากล่างไปบน ส่วนโครงสร้างข้อมูลที่มีลักษณะต้นไม้นั้นเราจะวาดหรือให้เจริญจากบนลงมาล่างดังรูปที่ 2.1 จุดที่มีการแตกกิ่งก้านสาขาออกไปจะเรียกว่า จุดแตกกิ่งหรือบัพ (Node) โดยข่าวสารจะเก็บอยู่ที่บัพ กิ่งที่ต่อระหว่างบัพ จะแสดงความสัมพันธ์ระหว่างบัพ เรียกว่า กิ่ง (Branch) อนึ่งตามรูปที่ 2.1 จุดปลายของกิ่งจะเรียกว่าบัพด้วย (สุชาย ธนวเสถียร และ วิชัย จิวังกูร, 2535)



รูปที่ 2.1 โครงสร้างข้อมูลแบบต้นไม้

นิสาชล โตอดิเทพย์ (2535) ได้นิยามโครงสร้างต้นไม้ไว้ดังนี้

ต้นไม้ประกอบด้วยสมาชิกที่เรียกว่า บัพ ที่

- ว่าง (ไม่มีบัพในต้นไม้) หรือ
- มีบัพเพียงอันเดียว คือ บัพราก (Root Node) หรือ
- มีบัพหนึ่งถือเป็นบัพราก ส่วนบัพที่เหลือแบ่งเป็นต้นไม้ย่อย T_1 , T_2 , ... , T_K (K >= 0) โดยมีคุณสมบัติเป็นต้นไม้เช่นกัน

ความสัมพันธ์ระหว่างบัพรากและบัพรากของต้นไม้ย่อยเป็นไปในลักษณะพ่อกับลูกคือบัพรากเป็นบัพพ่อ (Parent Node) และบัพรากของต้นไม้ย่อยเป็นบัพลูก (Child Node)

ตัวอักษรหรือตัวเลขในบัพเป็นชื่อของบัพหรืออาจจะเป็นค่าที่อยู่ในบัพ ต้นไม้ในรูป 2.1 มีบัพ A เป็นบัพราก ต้นไม้ย่อยของ A มี 3 ต้นคือ ต้นไม้ T_1 , T_2 และ T_3 ซึ่งมี B, C และ D เป็นบัพราก ตามลำดับ ต้นไม้ B มีต้นไม้ E และ F เป็นต้นไม้ย่อย ต้นไม้ C มีต้นไม้ G เป็นต้นไม้ย่อย ต้นไม้ D มี H และ I เป็นต้นไม้ย่อย

บัพพี่น้อง (Brother Node) คือ บัพที่มีพ่อเดียวกัน บัพ B, C และ D เป็นพี่น้องกัน บัพ E และ F เป็นพี่น้องกัน H และ I เป็นพี่น้องกัน

กิ่ง (Branch) หรือเอดจ์ (Edge) คือ เส้นที่เชื่อมต่อระหว่างพ่อกับลูก

บัพที่ไม่มีลูก เช่น บัพ E, F, G, H และ I เรียกว่า บัพใบ (Leaf Node)

ส่วนบัพที่ไม่ใช่บัพราก และไม่ใช่บัพใบ เช่น บัพ B, C และ D เรียกว่า บัพกิ่ง (Branch Node)

ดีกรี (Degree) ของบัพ X ใด ๆ คือจำนวนลูกของบัพนั้น เช่น ดีกรีของบัพใบเท่ากับ 0 ดีกรีของบัพ B เท่ากับ 2

บัพที่มาทีหลังทันที (Direct Descendant Node) ของบัพ X ใด ๆ คือ บัพที่เป็นลูกของบัพ X นั้น เช่น B, C และ D เป็นบัพที่มาทีหลังทันทีของบัพ A

บัพที่มาทีหลัง (Descendant Node) คือ บัพลูกของ X และบัพที่เป็นบัพที่มาทีหลังของลูกของ X เช่น บัพที่มาทีหลังของบัพ A คือ ทุกบัพที่เหลือในต้นไม้นี้

บัพที่มาก่อนทันที (Direct Ancestor Node) ของบัพ X ใด ๆ (ยกเว้นบัพราก) คือ บัพพ่อของ X นั้น เช่น บัพ E และ F มีบัพ B เป็นบัพที่มาก่อนทันที และ บัพ B มี A เป็นบัพที่มาก่อนทันที

บัพที่มาก่อน (Ancestor Node) ของบัพ X ใด ๆ คือ บัพพ่อและบัพที่เป็นบัพที่มาก่อนของบัพพ่อของ X เช่น A, B เป็นบัพที่มาก่อนของ E และ F

ระดับ (Level) เป็นหมายเลขแสดงระดับของบัพในต้นไม้ บัพรากถูกกำหนดให้อยู่ในระดับ 0 ส่วนบัพที่มาทีหลังทันทีอยู่ในระดับถัดไป คือ 1 คือถ้าบัพ X (ยกเว้นบัพใบ) ใด ๆ อยู่ในระดับ i ลูกของบัพ X จะอยู่ในระดับ i+1

ความสูงหรือความลึก (Height หรือ Depth) ของต้นไม้ คือ ระดับสูงสุดของต้นไม้นั้น จากรูปที่ 2.1 ต้นไม้นี้มีความสูงเท่ากับ 2

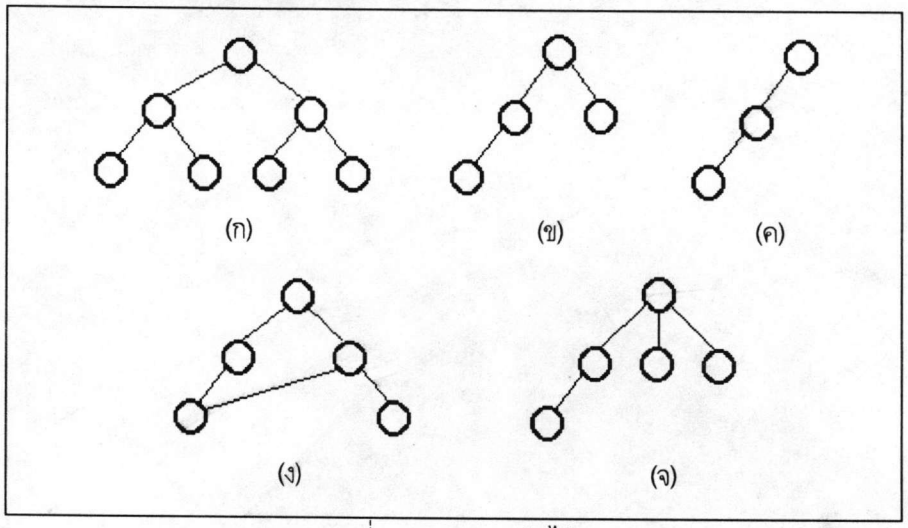
1. ต้นไม้ไบนารี (Binary Tree)

นิสาชล โตอดิเทพย์ (2535) ได้นิยามต้นไม้ไบนารีไว้ว่า เป็นต้นไม้ที่

1. ว่าง (คือ ไม่มีบัพใด ๆ) หรือ

2. มีบัพเพียงบัพเดียว คือ บัพราก หรือ

3. มีบัพหนึ่งเป็นบัพราก และส่วนที่เหลือแยกเป็นต้นไม้ย่อย 2 ต้น เรียกว่า ต้นไม้ย่อยทางซ้าย และต้นไม้ย่อยทางขวา โดยต้นไม้ย่อยทั้ง 2 นั้นมีคุณสมบัติเป็นต้นไม้ไบนารีเช่นกัน



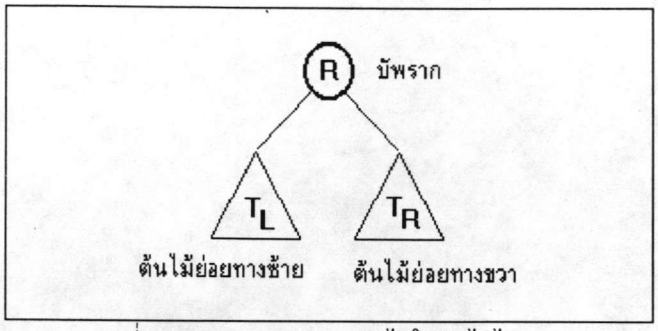
รูปที่ 2.2 ตัวอย่างต้นไม้

จากรูปที่ 2.2 (ก) , (ข) และ (ค) เป็นต้นไม้ไบนารี แต่ (ง) และ (จ) ไม่เป็นต้นไม้ไบนารี

2. การท่องเข้าไปในโครงสร้างต้นไม้ (Tree Traversal)

การนำต้นไม้ไบนารีมาใช้ประโยชน์นั้น หมายความว่า เราต้องมีวิธี ท่อง หรือ Traverse เข้าไปในต้นไม้อย่างมีแบบแผน เพื่อไป "เยี่ยม" บัพต่าง ๆ บัพละ 1 ครั้ง คำว่า เยี่ยม หมายถึง เราจะไปยังบัพเพื่อประมวลผลบางอย่างกับบัพนั้น เช่น หาข่าวสาร ดังนั้นผลลัพธ์จากการท่องเข้าไปในต้นไม้ คือ จะได้ข่าวสารที่เก็บโดยบัพเหล่านั้นออกมาในรูปเชิงเส้น ซึ่งนำไปใช้ประโยชน์ได้

สุชาย ธนวเสถียร และ วิชัย จิวังกูร (2535) กล่าวไว้ว่า ความคิดเริ่มต้นของการท่องก็คือ เราต้องเยี่ยมแต่ละบัพพร้อมทั้งต้นไม้ย่อยทางซ้ายและทางขวาของบัพนั้น ตามแนวคิดนี้ เราจะแยกต้นไม้ไบนารีออกเป็น 3 ส่วน คือ R, T_L และ T_R ตามรูปที่ 2.3 โดยที่ R จะแทนบัพราก T_L จะแทนต้นไม้ย่อยทางซ้าย และ T_R จะแทนต้นไม้ย่อยทางขวา ดังนั้นจะได้วิธีท่องอยู่ 6 วิธีคือ $R\ T_L\ T_R$, $T_L\ R\ T_R$, $T_L\ T_R\ R$, $T_R\ T_L\ R$, $T_R\ R\ T_L$ และ $R\ T_R\ T_L$



รูปที่ 2.3 แนวคิดการท่องเข้าไปในต้นไม้ไบนารี

ถึงแม้ว่าจะมีวิธีท่อง 6 วิธี แต่วิธีหลักคือ 3 วิธีแรก ซึ่ง 3 วิธีแรกมีชื่อเรียกและลักษณะรายละเอียดการท่องดังนี้

(1) ท่องแบบก่อนลำดับ (Pre-order Traversal) - R T_L T_R

- เยี่ยม R
- ท่องเข้าไปใน T_L (ของ R) แบบก่อนลำดับ
- ท่องเข้าไปใน T_R (ของ R) แบบก่อนลำดับ

(2) ท่องแบบตามลำดับ (In-order Traversal) - T_L R T_R

- ท่องเข้าไปใน T_L (ของ R) แบบตามลำดับ
- เยี่ยม R
- ท่องเข้าไปใน T_R (ของ R) แบบตามลำดับ

(3) ท่องแบบหลังลำดับ (Post-order Traversal) - T_L T_R R

- ท่องเข้าไปใน T_L (ของ R) แบบหลังลำดับ
- ท่องเข้าไปใน T_R (ของ R) แบบหลังลำดับ
- เยี่ยม R

ส่วนอีก 3 วิธี เกิดจากการสับเปลี่ยนตำแหน่ง T_R และ T_L ของ 3 วิธีที่กล่าวมาแล้ว ดังนั้น 3 วิธีหลังจะมีชื่อคล้าย ๆ กับ 3 วิธีแรก เพียงแต่มีคำว่า Reverse เพิ่มเข้าไปเท่านั้น ดังนี้

(4) แบบ Reverse Pre-order (R T_R T_L)

(5) แบบ Reverse In-order (T_R R T_L)

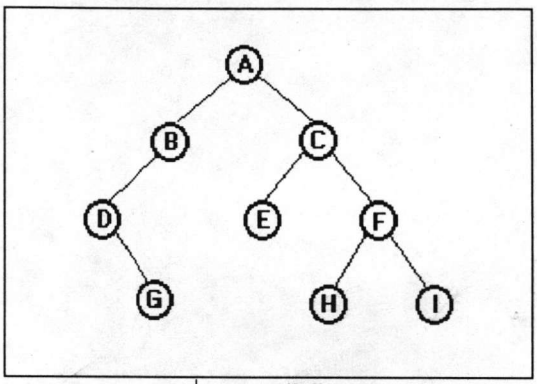
(6) แบบ Reverse Post-order (T_R T_L R)

2.1 การท่องแบบก่อนลำดับ (R T_L T_R)

สมมติมีต้นไม้ไบนารีตามรูปที่ 2.4 เราจะท่องเข้าไปในต้นไม้นี้อย่างก่อนลำดับ เพื่อต้องการนำค่าของทุกบัพออกมาพิมพ์

เริ่มที่บัพราก A แล้วให้เยี่ยมบัพ A ก่อน (จากนิยาม R T_L T_R) เราจะให้คำว่า เยี่ยม ของเรา หมายถึง พิมพ์ ชื่อบัพออกมา ต่อไปให้ท่องเข้าไปใน T_L ของบัพ A ไปอยู่ที่ B (เนื่องจาก B

เป็นบัพรากของต้นไม้ย่อยทางซ้ายของ A) เมื่อคิดอย่างเรียกซ้ำ (Recursive) เท่ากับเราตั้งต้นทำ R T_L T_R ใหม่ ดังนั้นบัพ B จะตรงกับตำแหน่ง R ใน R T_L T_R นั่นคือเราต้องเยี่ยมบัพ B เมื่อเยี่ยมบัพ B แล้ว เราก็จะท่องเข้าไปใน T_L ของบัพ B ไปอยู่ที่ D แล้วตั้งต้นทำ R T_L T_R ใหม่ ดังนั้นจึงเยี่ยมบัพ D จากนั้นท่องเข้าไปใน T_L ของบัพ D แต่เนื่องจาก T_L ของบัพ D ว่างเปล่า ดังนั้นจึงทำตำแหน่ง T_R ใน R T_L T_R ต่อไป T_R ของบัพ D คือ G จึงได้ค่า G พิมพ์ออกมา สรุปได้ว่า ในขณะนี้ชุดของบัพที่พิมพ์ออกมา คือ ABDG



รูปที่ 2.4 ต้นไม้ไบนารี

ต่อไปให้ท่องเข้าไปใน T_L ของ G ซึ่งว่างเปล่า ดังนั้นก็ข้ามไปท่องใน T_R ของ G ซึ่งก็ว่างเปล่าอีกเช่นกัน ณ จุดนี้เท่ากับว่าเราได้ท่องใน T_L ของบัพ A ครบแล้ว ดังนั้นจึงข้ามไปท่องใน T_R ของบัพ A นั่นคือไปตั้งต้นที่บัพ C แล้วเริ่มต้นทำการท่องตามนิยาม R T_L T_R ใหม่อีก โดยเริ่มเยี่ยมบัพ C แล้วท่องเข้าไปใน T_L ของบัพ C ต่อไปเริ่มเยี่ยมบัพ E จากนั้นก็ท่องเข้าไปใน T_L ของบัพ E ซึ่งว่างเปล่า ต่อไปก็ท่องเข้าไปใน T_R ของบัพ E ซึ่งก็ว่างเปล่าอีก ดังนั้นเท่ากับได้ท่องเข้าไปใน T_L ของบัพ C ครบแล้ว ต่อไปก็ท่องเข้าไปใน T_R ของบัพ C จนครบ ..... ในที่สุดก็เท่ากับว่าได้ท่องเข้าไปใน T_R ของบัพ A ครบแล้ว

ดังนั้นชุดข่าวสารที่พิมพ์ออกมา ก็จะเป็นดังนี้ ABDGCEFHI

2.2 การท่องแบบตามลำดับ (T_L R T_R)

การท่องแบบนี้มีลักษณะคล้ายกับการท่องแบบก่อนลำดับ นั่นคือต้องท่องเข้าไปในต้นไม้ย่อยทางซ้าย (T_L) ให้ครบก่อน แล้วจึงค่อยท่องเข้าไปในต้นไม้ย่อยทางขวาได้ การท่องแบบตามลำดับนี้เราจะเยี่ยมบัพรากภายหลังจากที่ได้ท่องเข้าไปใน T_L อย่างตามลำดับครบหมดแล้วเท่านั้น ต้นไม้ไบนารีตามรูปที่ 2.4 เมื่อท่องเข้าไปอย่างตามลำดับแล้วจะได้ชุดลำดับของบัพที่พิมพ์ออกมาเป็น DGBAECHFI

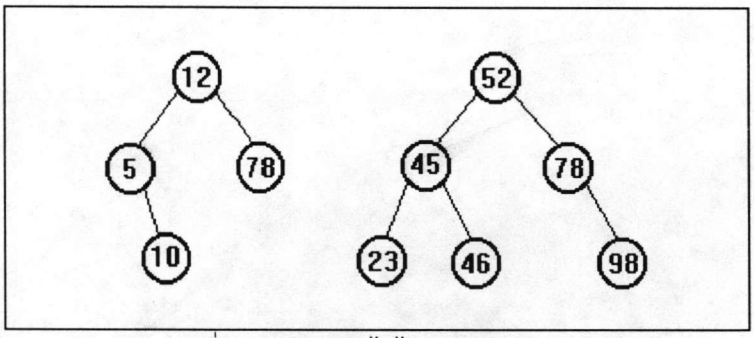
2.3 การท่องเข้าแบบหลังลำดับ (T_L T_R R)

การท่องแบบนี้จะต้องท่องเข้าไปในต้นไม้ย่อยทางซ้าย (T_L) ก่อน แล้วจึงค่อยท่องเข้าไปในต้นไม้ย่อยทางขวา (T_R) แล้วจึงเยี่ยมบัพรากภายหลังจากที่ได้ท่องเข้าไปใน T_L และ T_R ครบหมดแล้ว ต้นไม้ไบนารีตามรูปที่ 2.4 เมื่อท่องอย่างหลังลำดับแล้วจะได้ชุดลำดับดังนี้ DGHEBFCA

3. ต้นไม้ไบนารีค้นหาข้อมูล (Binary Search Tree)

Kruse, Leung และ Tondo (1991) ได้นิยามต้นไม้ไบนารีค้นหาข้อมูลไว้ว่า เป็นต้นไม้ไบนารีที่ว่างเปล่า หรือเป็นต้นไม้ไบนารีที่แต่ละบัพมีข้อมูลที่

1. ทุกข้อมูลในต้นไม้ย่อยทางซ้าย จะต้องน้อยกว่าข้อมูลในบัพราก
2. ถ้ามีต้นไม้ย่อยทางขวาด้วยแล้ว ค่าของข้อมูลในบัพราก จะต้องน้อยกว่าทุก ๆ ค่าในต้นไม้ย่อยทางขวา
3. ต้นไม้ย่อยทางซ้ายและต้นไม้ย่อยทางขวาของบัพราก จะต้องเป็นต้นไม้ไบนารีค้นหาข้อมูลเช่นกัน

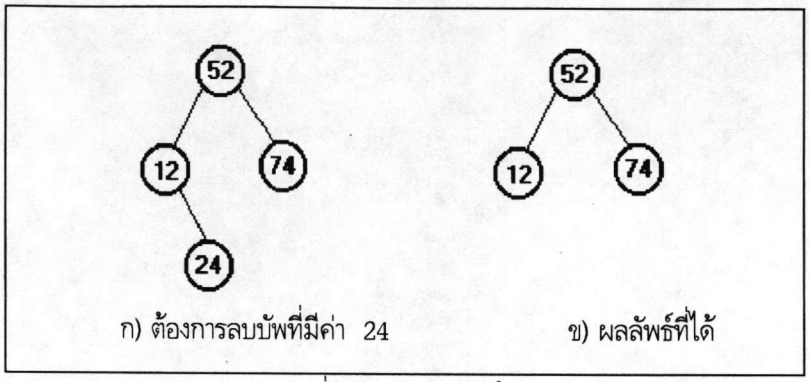


รูปที่ 2.5 แสดงต้นไม้ไบนารีค้นหาข้อมูล

3.1 การเพิ่มบัพเข้าไปในต้นไม้ไบนารีค้นหาข้อมูล ทำโดยให้ข้อมูลตัวแรกอยู่ในบัพราก จากนั้นก็ต่อเติมบัพต่าง ๆ โดยถือหลักว่า ถ้าข้อมูลที่เข้ามาใหม่ มีค่าน้อยกว่าบัพที่มีอยู่แล้ว ณ จุดนั้น ก็ให้ข้อมูลนั้นอยู่ในบัพลูกทางซ้ายของจุดนั้น หรือให้เลื่อนไปเปรียบเทียบกับบัพทางซ้ายที่มีอยู่แล้ว ในกรณีที่ข้อมูลใหม่มีค่ามากกว่าบัพที่มีอยู่แล้ว ณ จุดนั้น ก็ให้ข้อมูลนั้นอยู่ในบัพลูกทางขวาของจุดนั้น หรือให้เลื่อนไปเปรียบเทียบกับบัพทางขวาที่มีอยู่แล้ว (สุชาย ธนวเสถียร และ วิชัย จิวังกูร, 2535)

3.2 การลบบัพออกจากต้นไม้ไบนารีค้นหาข้อมูล มีความยากลำบากกว่าการเพิ่มบัพมาก เพราะอาจมีการลบบัพที่อยู่ภายในต้นไม้ ซึ่งก็ต้องทำให้บัพอื่น ๆ ที่เหลืออยู่ ยังคงมีความสัมพันธ์กันในลักษณะของต้นไม้ไบนารีค้นหาข้อมูล สามารถแยกพิจารณาเป็น 2 กรณีคือ

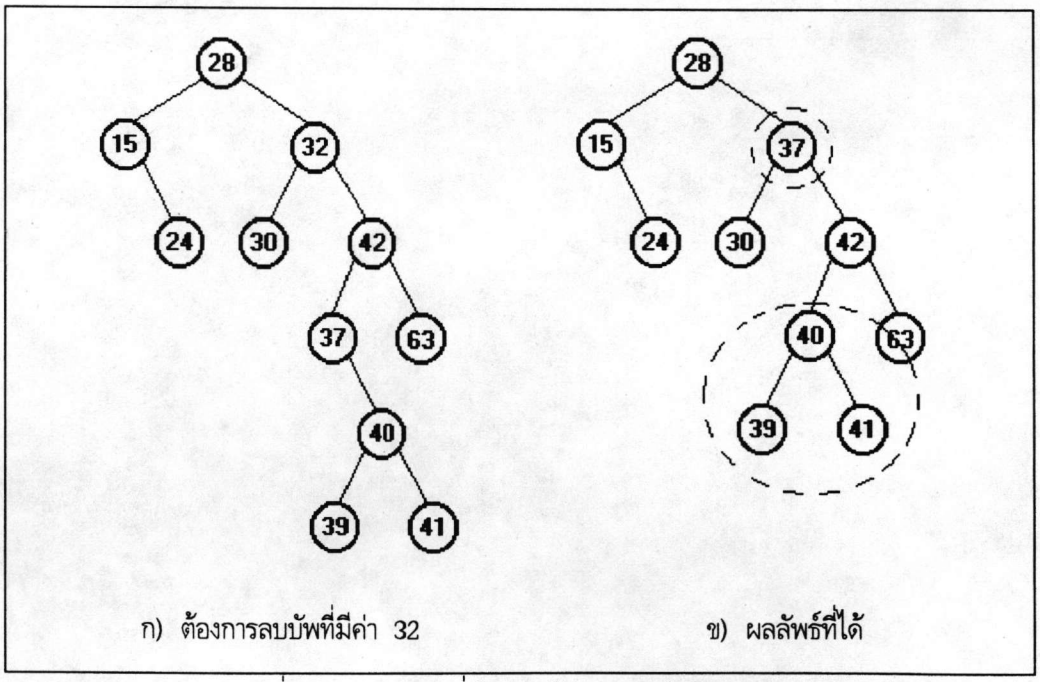
3.2.1 กรณีลบบัพใบ ก็สามารถลบบัพนั้นทิ้งไปได้เลย โดยไม่ต้องมีการเปลี่ยนแปลงบัพอื่น ๆ



ก) ต้องการลบบัพที่มีค่า 24 ข) ผลลัพธ์ที่ได้

รูปที่ 2.6 การลบบัพใบ

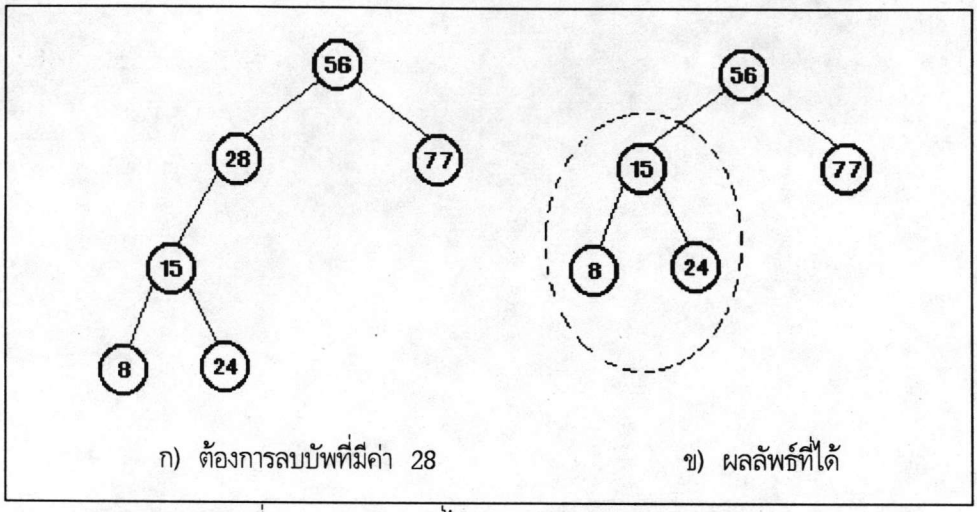
3.2.2 กรณีลบบัพที่มีลูกทางซ้าย และ/หรือ มีลูกทางขวา ให้พิจารณาว่าบัพที่ต้องการลบนั้นมีลูกทางขวาหรือไม่ ถ้ามีลูกทางขวา ก็ให้เพื่อนบ้านแบบตามลำดับที่อยู่ถัดไปมาอยู่แทนที่ ส่วนบัพที่อยู่ถัดจากเพื่อนบ้านลงไป ก็ให้เป็นลูกทางซ้ายของบัพพ่อของเพื่อนบ้านนั้น ซึ่งจะสังเกตได้ว่า บัพเพื่อนบ้านนั้นจะไม่มีลูกทางซ้ายเสมอ (สุชาย ธนวเสถียร และ วิชัย จิวังกูร, 2535)



ก) ต้องการลบบัพที่มีค่า 32 ข) ผลลัพธ์ที่ได้

รูปที่ 2.7 การลบบัพที่มีลูกทางซ้าย และ/หรือ ลูกทางขวา

ในกรณีที่บัพที่ต้องการลบไม่มีลูกทางขวา แต่มีลูกทางซ้าย ก็ให้ลบบัพนั้น แล้วนำต้นไม้ย่อยที่มีลูกทางซ้ายของบัพนั้นเป็นบัพรากมาแทนที่



ก) ต้องการลบบัพที่มีค่า 28 ข) ผลลัพธ์ที่ได้

รูปที่ 2.8 การลบบัพที่ไม่มีลูกทางขวา แต่มีลูกทางซ้าย

จากรูปที่ 2.8 หากบัพที่มีค่า 28 เป็นบัพราก ก็สามารถลบบัพนั้นได้เลยโดยไม่ต้องเปลี่ยนแปลงบัพอื่น ๆ

4. ต้นไม้ที่มีความสูงสมดุล (AVL Tree)

รูปร่างของต้นไม้ไบนารีค้นหาข้อมูลที่ดีนั้น จะต้องเป็นต้นไม้ที่มีความสูงของต้นไม้ย่อยทางซ้ายและทางขวาเท่ากัน จึงจะทำให้การค้นหาข้อมูลทุกครั้ง มีจำนวนครั้งของการเปรียบเทียบข้อมูลเท่า ๆ กันในกรณีที่ค้นหาข้อมูลนั้นไม่พบ แต่เนื่องจากการทำให้ต้นไม้มีความสูงของต้นไม้ย่อยทางซ้ายและทางขวาเท่ากัน หรือเป็นต้นไม้ที่มีความสูงสมดุลจริง ๆ นั้น สามารถทำได้ แต่ต้องใช้เวลามาก การทำให้ต้นไม้เกือบสมดุลจะง่ายกว่า โดยต้นไม้ที่เกือบสมดุลนั้น จะหมายถึง ต้นไม้ที่มีความสูงสมดุล (Height Balanced Tree) ในปี ค.ศ. 1962 นักคณิตศาสตร์ชาวรัสเซียสองคน ชื่อ G. M. **A**del'son - **V**el'skil และ E. M. **L**andis ได้นิยามต้นไม้ที่มีความสูงสมดุลไว้ว่า

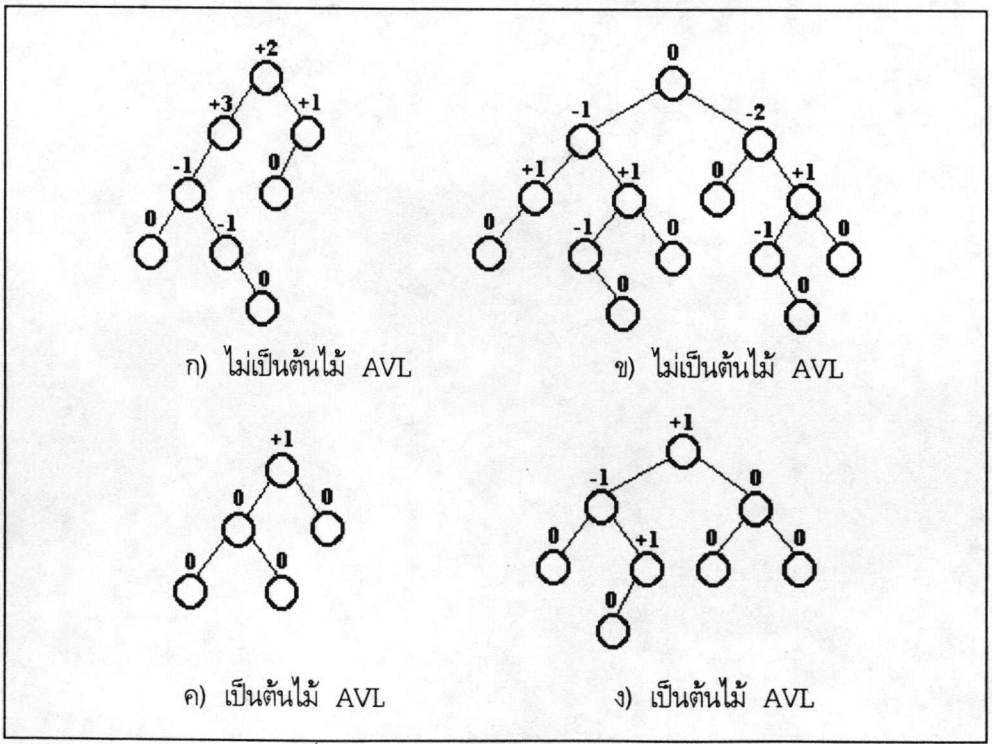
ต้นไม้ที่มีความสูงสมดุลเป็นต้นไม้ไบนารีค้นหาข้อมูล ที่ความสูงของต้นไม้ย่อยทางซ้ายและต้นไม้ย่อยทางขวาของบัพรากแตกต่างกันไม่เกิน 1 ระดับ และต้นไม้ย่อยทางซ้ายและต้นไม้ย่อยทางขวานั้นก็ต้องเป็นต้นไม้ที่มีความสูงสมดุลด้วยเช่นกัน

ต้นไม้ที่มีความสูงสมดุลมีอีกชื่อหนึ่งว่า **AVL** Tree ทั้งนี้เพื่อเป็นเกียรติแก่นักคณิตศาสตร์ชาวรัสเซียทั้งสองคน

ผลต่างของความสูงของต้นไม้ย่อยทางซ้ายและทางขวาของบัพรากใด ๆ จะเรียกว่า น้ำหนักของบัพนั้น โดยมีค่าเป็น 0, +1 หรือ -1 การคำนวณน้ำหนักสำหรับแต่ละบัพใช้หลักการดังนี้

น้ำหนักของบัพ R = ความสูงของต้นไม้ย่อยทางซ้ายของ R - ความสูงของต้นไม้ย่อยทางขวาของ R

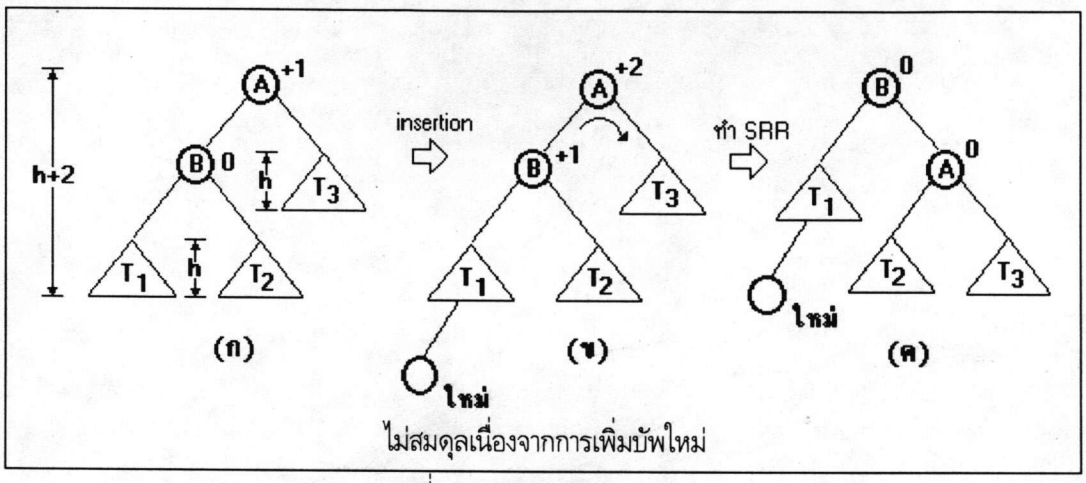
กรณีที่บัพ R ไม่มีลูกทั้งซ้ายและขวา น้ำหนักประจำบัพ R จะเท่ากับ 0



รูปที่ 2.9 ตัวอย่างการพิจารณาต้นไม้ AVL

4.1 หลักการทำต้นไม้ให้สมดุล สุชาย ธนวเสถียร และ วิชัย จิวังกูร (2535) ได้อธิบายถึงการทำต้นไม้ให้สมดุลไว้ว่า การทำต้นไม้ให้สมดุลนั้นมี 4 รูปแบบ ดังนี้

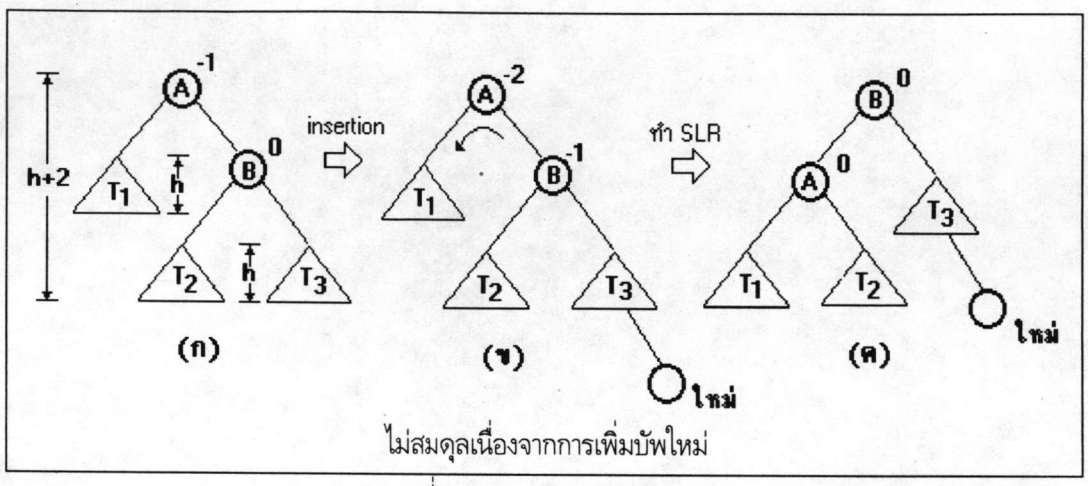
4.1.1 SRR (Single Right Rotation)



รูปที่ 2.10 แสดงการทำ SRR

จากรูปที่ 2.10 (ก) ความสูงของต้นไม้ที่มีบัพ A เป็นบัพรากเท่ากับ h + 2 และน้ำหนักของบัพ A เท่ากับ +1 ส่วนความสูงของต้นไม้ย่อยทางขวาและทางซ้ายของบัพ B เท่ากับ h และน้ำหนักของบัพ B เท่ากับ 0 เมื่อมีการเพิ่มบัพใหม่ที่ต้นไม้ย่อย T_1 (รูปที่ 2.10 (ข)) ซึ่งเป็นต้นไม้ย่อยทางซ้ายของบัพ B ทำให้น้ำหนักของบัพ A เปลี่ยนเป็น +2 เกิดต้นไม้ไม่สมดุลขึ้น จึงต้องทำการปรับต้นไม้ให้สมดุลตามรูปที่ 2.10 (ค) โดยการให้บัพ A และต้นไม้ย่อยทางขวาของบัพ A (T_3) ไปเป็นต้นไม้ย่อยทางขวาของบัพ B แล้วให้ต้นไม้ย่อยทางขวาของบัพ B เดิม (T_2) ไปเป็นต้นไม้ย่อยทางซ้ายของบัพ A ซึ่งจากรูปที่ 2.10 (ข) ไปเป็นรูปที่ 2.10 (ค) ก็คล้าย ๆ กับการหมุนต้นไม้ไปทางขวาหนึ่งครั้ง (Single Right Rotation) ตามทิศทางลูกศรของรูปที่ 2.10 (ข)

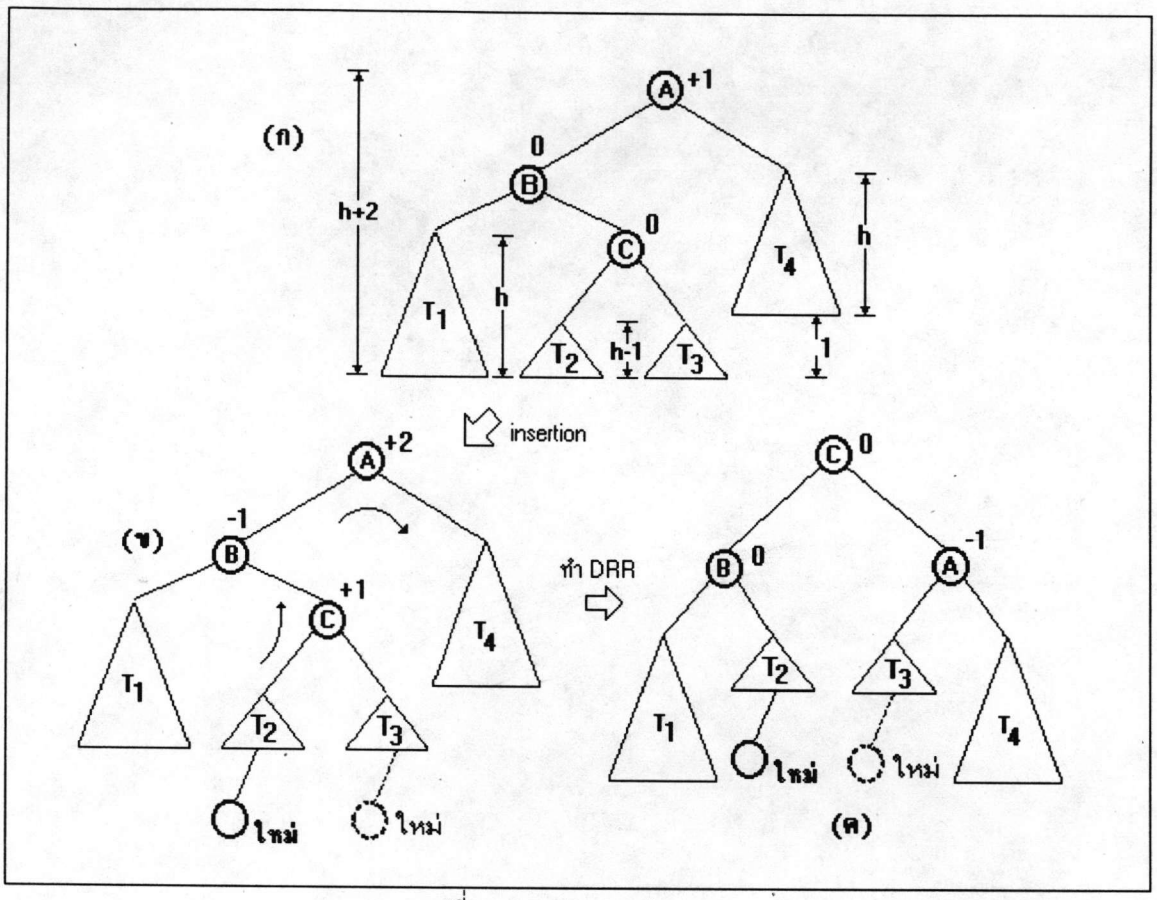
4.1.2 SLR (Single Left Rotation)



รูปที่ 2.11 แสดงการทำ SLR

จากรูปที่ 2.11 (ก) ความสูงของต้นไม้ที่มีบัพ A เป็นบัพรากเท่ากับ h + 2 และน้ำหนักของบัพ A เท่ากับ -1 ส่วนความสูงของต้นไม้ย่อยทางขวาและทางซ้ายของบัพ B เท่ากับ h และน้ำหนักของบัพ B เท่ากับ 0 เมื่อมีการเพิ่มบัพใหม่ที่ต้นไม้ย่อย T_3 (รูปที่ 2.11 (ข)) ซึ่งเป็นต้นไม้ย่อยทางขวาของบัพ B ทำให้น้ำหนักของบัพ A เปลี่ยนเป็น -2 จึงทำให้ต้นไม้ไม่สมดุล ต้องทำการปรับต้นไม้ให้สมดุลตามรูปที่ 2.11 (ค) โดยการให้บัพ A และต้นไม้ย่อยทางซ้ายของบัพ A (T_1) ไปเป็นต้นไม้ย่อยทางซ้ายของบัพ B แล้วให้ต้นไม้ย่อยทางซ้ายของบัพ B เดิม (T_2) ไปเป็นต้นไม้ย่อยทางขวาของบัพ A ซึ่งจากรูปที่ 2.11 (ข) ไปเป็นรูปที่ 2.11 (ค) ก็คล้าย ๆ กับการหมุนต้นไม้ไปทางซ้ายหนึ่งครั้ง (Single Left Rotation) ตามทิศทางลูกศรของรูปที่ 2.11 (ข)

4.1.3 DRR (Double Right Rotation)

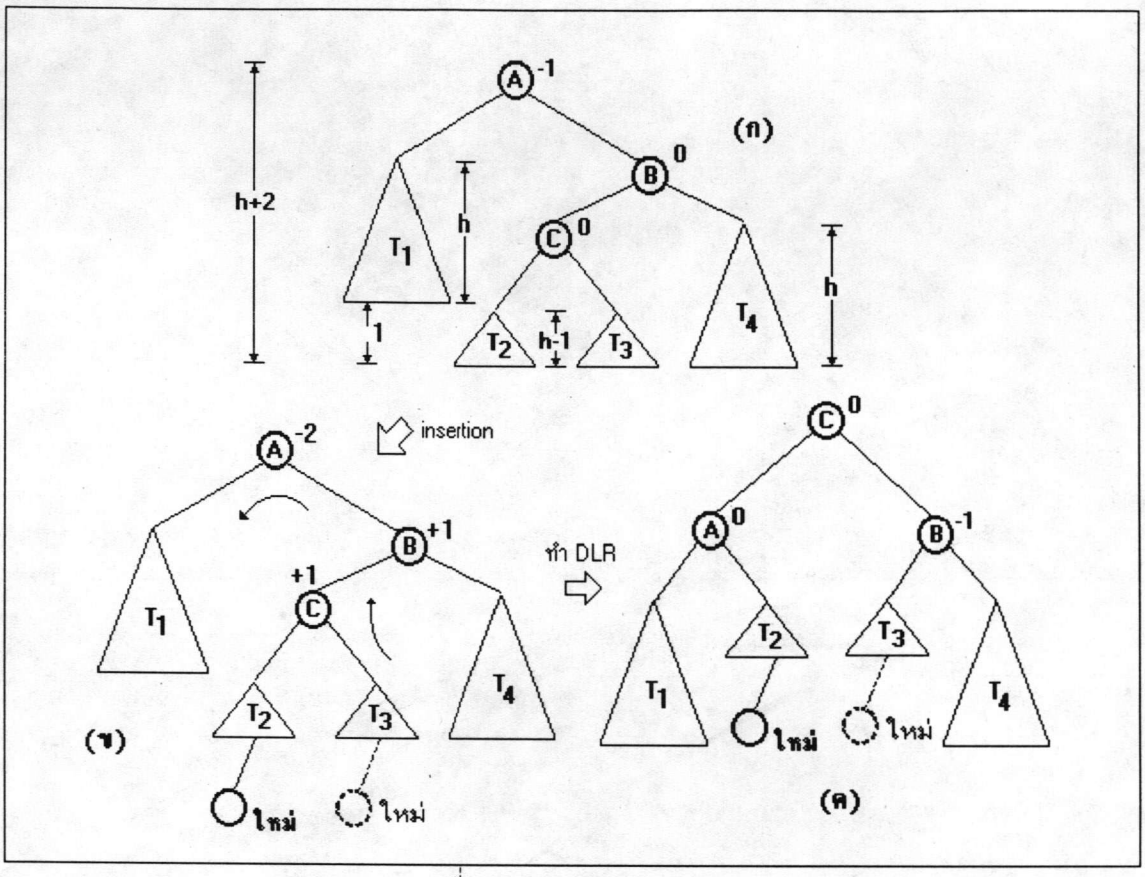


รูปที่ 2.12 แสดงการทำ DRR

ตามรูปที่ 2.12 (ข) เมื่อมีการเพิ่มบัพใหม่ที่ต้นไม้ย่อย T_2 ทำให้น้ำหนักบัพ A เท่ากับ +2 เกิดต้นไม้ไม่สมดุล จึงต้องทำการปรับต้นไม้ โดยให้บัพ A และต้นไม้ย่อยทางขวาของบัพ A (T_4) ไปเป็นต้นไม้ย่อยทางขวาของบัพ C แล้วให้ต้นไม้ย่อยทางขวาของบัพ C เดิม (T_3) ไปเป็นต้นไม้ย่อยทางซ้ายของบัพ A จากนั้นให้บัพ B และต้นไม้ย่อยทางซ้ายของบัพ B (T_1) ไปเป็นต้นไม้ย่อยทางซ้ายของบัพ C แล้วให้ต้นไม้ย่อยทางซ้ายของบัพ C เดิม (T_2) ไปเป็นต้นไม้ย่อยทางขวาของบัพ B ซึ่งจากรูปที่ 2.12 (ข) ไปเป็นรูปที่ 2.12 (ค) ก็คล้าย ๆ กับการหมุนต้นไม้ไปทางขวาก่อนหนึ่งครั้ง จากนั้นจึงหมุนทางซ้ายอีกหนึ่งครั้ง (Double Right Rotation) ตามทิศทางลูกศรของรูปที่ 2.12 (ข)

บัพใหม่ที่เป็นเส้นประนั้นหมายถึง อาจมีการเพิ่มบัพที่ต้นไม้ย่อย T_3 ก็ได้ หากมีการเพิ่มที่ต้นไม้ย่อยนี้ น้ำหนักของบัพ C ที่รูป 2.12 (ข) จะเท่ากับ -1 แทนที่จะเป็น +1 ส่วนน้ำหนักที่บัพ B และบัพ A ยังคงเท่ากับ -1 และ +2 ตามลำดับเหมือนเดิม จากนั้นก็ทำการปรับต้นไม้ด้วยวิธี DRR เช่นเดิม แต่จะได้บัพ A ที่รูป 2.12 (ค) มีน้ำหนักเป็น 0 และบัพ B มีน้ำหนักเป็น -1 แทน

4.1.4 DLR (Double Left Rotation)

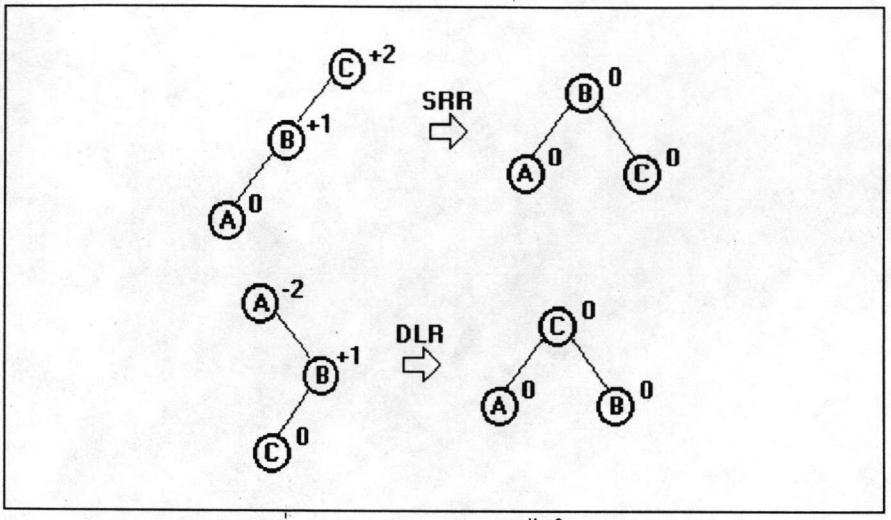


รูปที่ 2.13 แสดงการทำ DLR

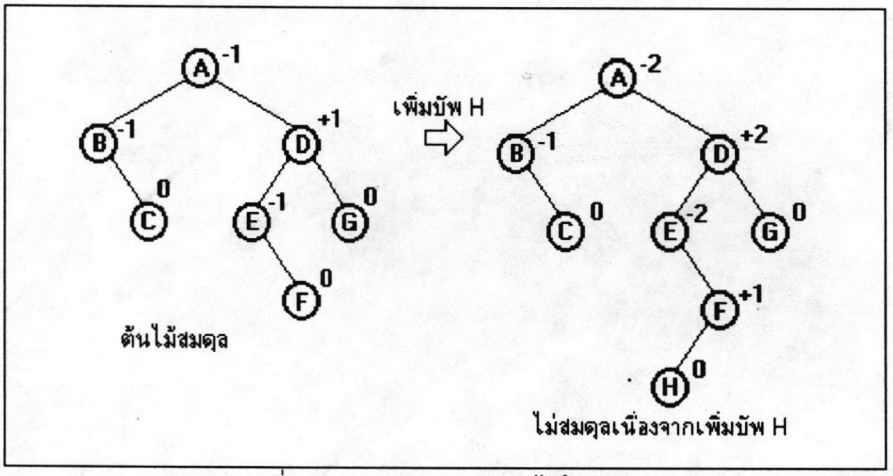
ตามรูปที่ 2.13 (ข) เมื่อมีการเพิ่มบัพใหม่ที่ต้นไม้ย่อย T_2 ทำให้น้ำหนักบัพ A เท่ากับ -2 เกิดต้นไม้ไม่สมดุล จึงต้องทำการปรับต้นไม้ให้สมดุล โดยให้บัพ A และต้นไม้ย่อยทางซ้ายของบัพ A (T_1) ไปเป็นต้นไม้ย่อยทางซ้ายของบัพ C แล้วให้ต้นไม้ย่อยทางซ้ายของบัพ C เดิม (T_2) ไปเป็นต้นไม้ย่อยทางขวาของบัพ A จากนั้นให้บัพ B และต้นไม้ย่อยทางขวาของบัพ B (T_4) ไปเป็นต้นไม้ย่อยทางขวาของบัพ C แล้วให้ต้นไม้ย่อยทางขวาของบัพ C เดิม (T_3) ไปเป็นต้นไม้ย่อยทางซ้ายของบัพ B ซึ่งจากรูปที่ 2.13 (ข) ไปเป็นรูปที่ 2.13 (ค) ก็คล้าย ๆ กับการหมุนต้นไม้ไปทางซ้ายก่อนหนึ่งครั้ง จากนั้นจึงหมุนทางขวาอีกหนึ่งครั้ง (Double Left Rotation) ตามทิศทางลูกศรของรูปที่ 2.13 (ข)

บัพใหม่ที่เป็นเส้นประนั้นหมายถึง อาจมีการเพิ่มบัพที่ต้นไม้ย่อย T_3 ก็ได้ หากมีการเพิ่มที่ต้นไม้ย่อยนี้ น้ำหนักของบัพ C ที่รูป 2.13 (ข) จะเท่ากับ -1 แทนที่จะเป็น +1 ส่วนน้ำหนักที่บัพ B และบัพ A ยังคงเท่ากับ +1 และ -2 ตามลำดับเหมือนเดิม จากนั้นก็ทำการปรับต้นไม้ด้วยวิธี DLR เช่นเดิม แต่จะได้บัพ A ที่รูป 2.13 (ค) มีน้ำหนักเป็น -1 และบัพ B มีน้ำหนักเป็น 0 แทน

SRR , SLR , DRR และ DLR นั้นไม่จำเป็นจะต้องเกิดขึ้นที่บัพรากเสมอไป ดังนั้น จากรูปที่ 2.10 (ก), 2.11 (ก), 2.12 (ก) และ 2.13 (ก) บัพ A จึงอาจเป็นบัพใด ๆ ในต้นไม้ที่มีความสูงสมดุลก็ได้

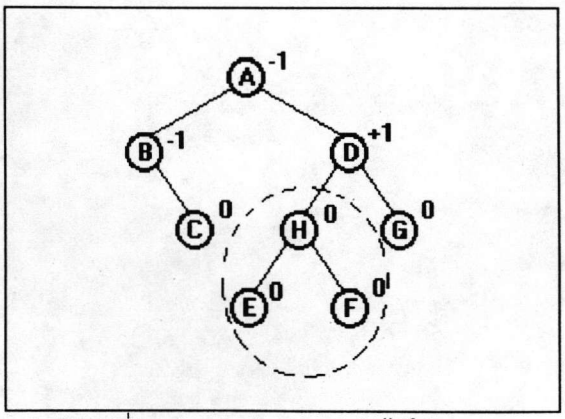


รูปที่ 2.14 ตัวอย่างการทำต้นไม้ให้สมดุล



รูปที่ 2.15 ตัวอย่างการทำต้นไม้ให้สมดุล

จากรูปที่ 2.15 หลังจากเพิ่มบัพ H ที่ตำแหน่งลูกทางซ้ายของบัพ F แล้ว จะได้ต้นไม้ไม่สมดุล เมื่อเรามองจากล่างขึ้นบนจะเห็นได้ว่า บัพที่ไม่สมดุล (ที่อยู่ล่างสุด) คือ บัพ E ดังนั้นการปรับต้นไม้ให้สมดุลก็เพียงกระทำที่ต้นไม้ย่อยที่มี E เป็นบัพรากด้วยวิธี DLR ส่วนบัพอื่น ๆ ไม่จำเป็นต้องปรับ ดังรูปที่ 2.16



รูปที่ 2.16 ตัวอย่างการทำต้นไม้ให้สมดุล

สภาพแวดล้อมเพื่อพัฒนาการโปรแกรมแบบวิชวล (Visual Programming)

การเชื่อมประสานกับผู้ใช้โดยใช้รูปภาพ (Graphic User Interface) ได้เริ่มมีบทบาทมากขึ้นเมื่อไมโครซอฟต์วินโดวส์ รุ่น 3.0 ประสบความสำเร็จทางด้านการตลาด สิ่งเหล่านี้ได้ช่วยให้ผู้ใช้ทั่วไป มีความสุขในการใช้คอมพิวเตอร์มากขึ้น อย่างไรก็ตาม สิ่งนี้ทำให้นักเขียนโปรแกรมมีความลำบากมากขึ้นในการพัฒนาโปรแกรม โดยเฉพาะเรื่องการที่จะต้องจดจำและปฏิบัติตามกฎเกณฑ์ของ Windows Application Program Interface หรือที่เราเรียกกันสั้น ๆ ว่า API เรื่องนี้นักเขียนโปรแกรมที่ใช้โปรแกรมอย่าง Software Development Kit (SDK) คงจะรับรู้ถึงความรู้สึกลำบากในสิ่งเหล่านี้ได้เป็นอย่างดี สิ่งเหล่านี้ได้ผลักดันให้บริษัทผู้ผลิตโปรแกรมใช้งานพยายามค้นหาวิธีการเขียนโปรแกรมใหม่บนสภาพแวดล้อมใหม่ ผลที่ได้ก็คือการเขียนโปรแกรมในลักษณะที่เป็นวิชวล (Visual Programming) ซึ่งเป็นการใช้ความก้าวหน้าของเทคโนโลยีการเขียนโปรแกรมแบบใหม่ที่เป็นทั้ง ออบเจกต์ (Object) และวิชวล (Visual) รวมเข้าด้วยกัน ปัจจุบันนี้เครื่องไม้เครื่องมือเพื่อใช้ในการเขียนโปรแกรมวิชวลได้มีออกสู่ตลาดแล้วมากมาย (PetZold, 1992)

ในอดีต การพัฒนาโปรแกรมบนคอมพิวเตอร์ส่วนบุคคล ไม่ว่าจะใช้ภาษา C , Pascal , Ada ฯลฯ จะกระทำโดยผ่านตัวแปลโปรแกรม (Compiler) โดยส่วนใหญ่ ซึ่งตัวแปลโปรแกรมจะมีหน้าที่อ่านแฟ้มข้อมูลของโปรแกรมดิบ (Source Code) ที่ป้อนเข้ามาจากนั้นก็ทำการสับซอยย่อยข้อความต่าง ๆ ในแฟ้มดังกล่าวเพื่อทำความเข้าใจ และแปลมันออกมาเป็นภาษาเครื่อง ซึ่งเป็นวิธีการดั้งเดิมที่กระทำซ้ำ ๆ กันมาตลอด 15 - 20 ปีล่วงมาแล้ว

จริงอยู่ที่โปรแกรมใช้งานประเภทตัวแปลโปรแกรมได้มีการพัฒนา และปรับปรุงอยู่ตลอดเวลา แต่ที่เห็นเป็นชิ้นเป็นอัน และดูเป็นการเปลี่ยนแปลงที่เห็นได้ชัดเกิดขึ้นเมื่อบริษัทบอร์แลนด์ (Borland) ได้สร้าง "สภาพแวดล้อมรวมเพื่อการพัฒนาโปรแกรม" หรือ IDE (Integrated Development Environment) เป็นบริษัทแรก โดยได้สร้างรวมไว้กับ "Turbo Pascal" ในปี ค.ศ. 1983

สภาพแวดล้อมรวมเพื่อการพัฒนาโปรแกรมของบริษัทบอร์แลนด์นั้น จะเป็นการ "รวม" เอาโปรแกรมบรรณาธิการ (Editor) , ตัวแปลโปรแกรม, และโปรแกรมแก้ไขจุดบกพร่องของอีกโปรแกรม (Debugger) เข้าไว้ที่ตัวโปรแกรมใช้งาน ทำให้การพัฒนาโปรแกรมสามารถกระทำได้อย่างรวดเร็ว สามารถแปลโปรแกรมในหน่วยความจำได้ทันทีที่เขียนโปรแกรมเสร็จ และสามารถตรวจสอบข้อผิดพลาด ตรวจสอบว่าบรรทัดไหนของโปรแกรมที่ผิดพลาดและผิดพลาดในเรื่องใด สุดท้ายแล้ว IDE ก็เลยกลายเป็นมาตรฐานหนึ่งของเครื่องมือพัฒนาโปรแกรม และทุกบริษัทผู้ผลิตเครื่องมือพัฒนาโปรแกรมก็เลือกที่จะเดินในเส้นทางดังกล่าวกันหมดทั้งนี้เพราะเป็นสิ่งที่ผู้ใช้ชื่นชอบมาก แต่การพัฒนาเครื่องมือพัฒนาโปรแกรมก็มาถึงจุดตันเมื่อถึงยุค "การเชื่อมประสานกับผู้ใช้โดยใช้รูปภาพ" ซึ่งบรรดาผู้พัฒนาโปรแกรมระดับทั่วไป รู้สึกยุ่งยากในการเขียนโปรแกรมติดต่อกับ API ของระบบปฏิบัติการ (Operating System)

แนวความคิดใหม่ในการสร้างสภาพแวดล้อมเพื่อการโปรแกรม หรือการสร้างภาษาใหม่เพื่อโปรแกรมจึงได้เกิดขึ้น เป็นการรวมเอา Windows-based IDE รวมเข้ากับโปรแกรมออกแบบการเชื่อมประสาน

(Interface) มีการเพิ่มขีดความสามารถเชิงการรวมโปรแกรมภาษาเครื่อง (Object Program) เข้าด้วยกัน และเพิ่มความสามารถการโต้ตอบเชิงการเชื่อมประสานกับผู้ใช้ (User Interface) มากขึ้น ผลลัพธ์ที่ออกมาก็ได้เป็นสภาพแวดล้อมเพื่อการโปรแกรมชนิดใหม่ ซึ่งเรียกว่า VDE หรือ Visual Development Environment โดยข้อดีของมันก็คือ ประการแรก เป็นโปรแกรมใช้งานสร้างโปรแกรมประยุกต์ที่ให้เราสามารถออกแบบ และเขียนโปรแกรมระบบการเชื่อมประสานได้โดยง่าย ประการที่สอง ผู้พัฒนาโปรแกรมไม่จำเป็นต้องจดจำ สนใจ หรือรู้จักกับ API ก็ได้ โดยหลักการแล้ว VDE จะเป็นประโยชน์อย่างมากในการพัฒนาโปรแกรม โดยมันจะลดความยุ่งยากในการเขียนโปรแกรมได้มาก ไม่ว่าจะเป็นเรื่องการเขียนโปรแกรมสร้างช่องรับข้อความ หรือการสร้างรายการเลือกแบบดึงลง (Pulldown Menu) หรืออื่น ๆ (Petzold, 1992)

<u>เชิงวัตถุ</u> (Object-Oriented)

วัตถุ (Object) คือ ปริมาณหนึ่งในระบบที่ประกอบขึ้นด้วยองค์ประกอบ 2 ส่วน คือ ข้อมูล และรหัสโปรแกรม ส่วนของข้อมูลใช้เก็บสถานะของตัวมันเอง เรียกว่า ดาต้า (Data) และส่วนของรหัสโปรแกรมที่ใช้ในการตอบสนองต่อวัตถุตัวอื่นในระบบเดียวกัน เรียกว่า วิธี (Method) วัตถุใด ๆ ในระบบจะสื่อสารกับวัตถุอื่น เพื่อให้บรรลุความต้องการของตน การสื่อสารนี้เป็นลักษณะ "ร้องขอและตอบสนอง" เมื่อวัตถุหนึ่งติดต่อกับอีกวัตถุหนึ่ง เราเรียกว่า มันกำลังส่งข้อความ (Message) ไปยังวัตถุอื่น

คำว่า คลาส (Class) กับ วัตถุ เป็นสิ่งคู่กัน เพราะคลาสคือเบ้าหลอมของวัตถุ เราจะดูว่าวัตถุนี้มีลักษณะ (Characteristic) อย่างไรให้ดูที่คลาส โดยที่ลักษณะเฉพาะตัวของวัตถุจะเรียกว่า Instance Variable และ Class Variable หมายถึง ลักษณะเฉพาะร่วมของคลาส

หลักการของ Object-oriented จะอาศัยการถ่ายทอดคุณสมบัติ (Inheritance) เป็นหลัก คลาสแต่ละคลาสในระบบสามารถให้กำเนิดลูกหลานได้ เรียกทายาทของคลาสว่า ซับคลาส (Subclass) ซับคลาสจะรับเอาคุณสมบัติของคลาสผู้ให้กำเนิดที่เรียกว่า Parent Class ซึ่งทำให้เกิดข้อดีคือ

1. การช่วยลดเวลาในการพัฒนาระบบ
2. ลดค่าใช้จ่ายในการพัฒนา
3. ได้ระบบที่มีโครงสร้างเป็นระเบียบและปรับปรุงเปลี่ยนแปลงได้ง่าย

เมื่อมีการสืบสายคุณสมบัติของวัตถุมาเป็นลำดับแล้ว ความสัมพันธ์ระหว่างวัตถุจะชัดเจนยิ่งขึ้น ยิ่งความสัมพันธ์ชัดมากขึ้น ก็มีผลให้การออกแบบระบบง่ายขึ้น และจะได้ระบบที่ซับซ้อนมากขึ้นเรื่อย ๆ ผู้ออกแบบสามารถถ่อกับอีกวัตถุหนึ่ง เราเรียกว่า มันกำลังส่งข้อความ (Message) ไปยังวัตถุอื่นยนักออกแบบคนอื่น ๆ (วรชัย เชาว์วีระประสิทธิ์, 2535)

นอกจากนี้ยังมีหลักการที่เรียกว่า Encapsulation และ Polymorphism

Encapsulation คือ กระบวนการปกปิดความลับของวัตถุ การเข้าถึงค่าสถานะภายในวัตถุจะกระทำได้โดยผ่านการเห็นชอบจากวัตถุเจ้าของเสียก่อน หมายความว่า การเปลี่ยนแปลงแก้ไขค่าตัวแปรภายในวัตถุจะ

ต้องกระทำผ่านวิธีของวัตถุเท่านั้น เช่นเดียวกับการไขก๊อกน้ำ คือหากเราไม่บิดก๊อก น้ำย่อมไม่ไหลออกมาแน่นอน (วรชัย เชาว์วีระประสิทธิ์, 2535)

ส่วน Polymorphism หมายถึง การบอกให้ทำเพียงอย่างเดียว แต่ได้รับการตอบสนองได้หลายแบบ หรือการส่งข้อความแบบเดียวกันไปยังวัตถุอื่นสามารถได้รับการตอบสนองได้หลายแบบ และไม่จำเป็นต้องได้รับการตอบสนองเหมือนกัน ขึ้นอยู่กับวัตถุที่รับข้อความเป็นสำคัญ โดยวัตถุจะใช้วิธีของตนเองในการตอบสนอง เนื่องจากระบบแบบเชิงวัตถุนั้นประกอบไปด้วยคลาสต่าง ๆ มากมาย โอกาสที่จะมีการกำหนดชื่อวิธีซ้ำกันได้มาก จึงต้องมีหลักการของ Polymorphism เพื่อที่ผู้พัฒนาระบบจะได้ไม่ต้องเสียเวลาตรวจสอบชื่อของวิธีในทุก ๆ คลาส เพื่อป้องกันการตั้งชื่อซ้ำก่อนจะทำการประกาศชื่อวิธีของตนเอง
(วรชัย เชาว์วีระประสิทธิ์, 2535)

<u>คลังชุดคำสั่งเชื่อมโยงแบบพลวัต</u> (Dynamic Link Library หรือ DLL)

DLL ถือเป็นหัวใจสำคัญของการทำงานร่วมกันภายใต้วินโดวส์ ชุดคำสั่ง , ข้อมูล และทรัพยากรต่าง ๆ โดยมากมักจะเก็บอยู่ในแฟ้มแบบ DLL

แฟ้ม DLL เป็นลักษณะพิเศษอย่างหนึ่งของแฟ้มกระทำการ (EXECUTABLE FILE) ถึงแม้ว่ามักจะมีสกุลเป็น .FON , .DLL , .DRV หรือ .SYS ตามชนิดของทรัพยากรใน DLL นั้น ๆ ก็ตาม แฟ้มสกุล .FON เป็นทรัพยากรอักขระ แฟ้มสกุล .DRV เป็นตัวควบคุมอุปกรณ์ (DEVICE DRIVER) แฟ้มสกุล .SYS เป็นแฟ้มของระบบวินโดวส์ที่มักจะเป็นตัวควบคุมอุปกรณ์บางอย่างด้วย และมี DLL บางอย่างที่ใช้สกุล .EXE ด้วยเหมือนกัน ซึ่งก็จะประกอบไปด้วยชุดคำสั่ง , ข้อมูล และทรัพยากรเหมือนกับโปรแกรมประยุกต์สำหรับวินโดวส์ทั่วไป (Rector, 1992)

เราไม่ได้สั่งดำเนินงาน DLL โดยตรง แต่เราสั่งดำเนินงานกับโปรแกรมประยุกต์ แล้วโปรแกรมประยุกต์นั้นอาจจะเรียกใช้ฟังก์ชั่นใน DLL เรียกค้นข้อมูลจาก DLL และใช้ทรัพยากรของ DLL เมื่อโปรแกรมประยุกต์มีการเรียกใช้ฟังก์ชั่นใน DLL ตัว DLL เองก็สามารถเรียกใช้ฟังก์ชั่น , ค้นหาข้อมูล และใช้ทรัพยากรของ DLL อื่นได้ด้วย

DLL ยังสามารถเรียกใช้ฟังก์ชั่นของโปรแกรมประยุกต์ได้ ถ้าได้มีการประกาศให้ฟังก์ชั่นนั้นเป็นฟังก์ชั่นแบบเรียกกลับได้ (CALLBACK) ซึ่งก็เหมือนกับการประกาศให้ฟังก์ชั่นนั้นสามารถเรียกใช้โดยวินโดวส์ได้ (EXPORT)

DLL จะถูกเรียกใช้งานทั้งโดยทางตรงและทางอ้อมจากโปรแกรมประยุกต์ ตัว DLL จะมีทรัพยากรต่าง ๆ ที่ทั้งวินโดวส์เองและโปรแกรมประยุกต์ต้องการใช้งาน และต้องใช้งานร่วมกันด้วย ดังนั้น DLL แบบหนึ่งภายในระบบของวินโดวส์ ก็จะมีเพียงสำเนาเดียวเท่านั้น ตัวอย่างง่าย ๆ ของ DLL ก็เช่น ฟังก์ชั่น GetMessage( ) ที่ใช้ในวงวนข้อความ (Message Loop) นั้นจะอยู่ในแฟ้ม DLL ชื่อ USER.EXE ฟังก์ชั่น Rectangle( ) ที่ใช้วาดรูปสี่เหลี่ยมก็อยู่ในแฟ้ม DLL ชื่อ GDI.EXE (Rector, 1992)

เราคุ้นเคยกับการเชื่อมโยงแบบสถิต (STATIC LINKING) ซึ่งเป็นวิธีแก้ปัญหาของการเรียกใช้ฟังก์ชั่นอื่นที่ไม่มีในโปรแกรม ตัวเชื่อมโยง (Linker) จะค้นหาตัวฟังก์ชั่นจากคลังชุดคำสั่ง (LIBRARY) ต่าง ๆ ว่ามีหรือไม่ ถ้ามี ก็จะเชื่อมโยงสำเนาของฟังก์ชั่นนั้นเข้ากับแฟ้มกระทำการ (สกุล .EXE) ซึ่งทำให้ภายในแฟ้มกระทำการมีสำเนาของฟังก์ชั่นนั้นอยู่ สมมติมีการใช้ฟังก์ชั่น strcpy( ) ในโปรแกรมต่าง ๆ เมื่อมีการสั่งดำเนินงานโปรแกรมเหล่านั้นพร้อม ๆ กัน ก็จะมีสำเนาของฟังก์ชั่น strcpy( ) มากมายอยู่ภายในหน่วยความจำ (Rector, 1992)

DLL มีลักษณะการเชื่อมโยงที่แตกต่างออกไป โดยวินโดวส์จะทำการเชื่อมโยงในขณะนำโปรแกรมเข้าสู่หน่วยความจำ หากโปรแกรมมีการเรียกใช้ฟังก์ชั่นของ DLL ก็จะมีระเบียนนำเข้า (Import Record) ของ DLL นั้นอยู่ในโปรแกรมด้วย โดยระเบียนนำเข้าจะเป็นชื่อของแฟ้ม DLL ที่มีฟังก์ชั่นนั้นอยู่ และมีหมายเลขลำดับของฟังก์ชั่นนั้นใน DLL ด้วย ซึ่งตัววินโดวส์จะค้นหาฟังก์ชั่นที่ต้องการจากคลังชุดคำสั่งแบบพลวัตต่าง ๆ ด้วยระเบียนนำเข้านี้ หากยังไม่มีฟังก์ชั่นนั้นอยู่ในหน่วยความจำ ก็ต้องนำฟังก์ชั่นนั้นเข้าสู่หน่วยความจำก่อน แล้วจึงค่อยเชื่อมโยงเข้ากับโปรแกรมประยุกต์อีกที การเชื่อมโยงแบบสถิตจะทำเพียงครั้งเดียวในตอนที่ทำการเชื่อมโยงโปรแกรมก่อนการดำเนินงานจริง ส่วนการเชื่อมโยงแบบพลวัตจะทำทุกครั้งที่มีการสั่งดำเนินงานโปรแกรม