บทที่ 2

แนวคิดและทฤษฎีสำคัญ

โครงสร้างข้อมูล

คำว่า "โครงสร้างข้อมูล" (Data Structures) เกิดจากคำสองคำ คือ "โครงสร้าง" และ "ข้อมูล" ซึ่งคำว่า "โครงสร้าง" นั้นเป็นความสัมพันธ์ระหว่างสมาชิกในกลุ่ม ดังนั้นโครงสร้างข้อมูล จึงหมายถึง ความสัมพันธ์ระหว่างข้อมูลที่อยู่ในโครงสร้างนั้น ๆ (นิสาชล โตอดิเทพย์, 2535) ข้อมูลเป็นสิ่งพื้นฐานที่ใช้ในการประมวลผลทางคอมพิวเตอร์ ดังนั้นการศึกษาถึงโครงสร้างข้อมูล จึงมีความสำคัญอย่างมากในศาสตร์คอมพิวเตอร์ โครงสร้างข้อมูลถ้าแบ่งตามภาพที่มองเห็นได้ จะแบ่งได้เป็น 2 ประเภทใหญ่ คือ

1. โครงสร้างข้อมูลแบบเชิงเส้น (Linear) ได้แก่โครงสร้างเชิงรายการ (List), สายอักขระ(String), กองซ้อน(Stack), และแถวคอย(Queue) เป็นต้น

2. โครงสร้างข้อมูลแบบไม่เป็นเชิงเส้น (Non Linear) ได้แก่ ต้นไม้(Tree), กราฟ และฮีป (Heap) เป็นต้น

โครงสร้างข้อมูลเหล่านี้ ได้ถูกนำมาใช้ในซอฟต์แวร์ระบบ (System Software) ด้วย เช่นโครงสร้างเชิงรายการใช้จัดการกับหน่วยความจำ แถวคอยใช้จัดลำดับของงานในระบบ ต้นไม้ใช้เป็นสารบบ (Directory) ของจานแม่เหล็ก (Disk) และการนำฮีปมาประยุกต์ใช้เป็นแถวคอยบุริมภาพ เป็นต้น

ก่อนที่จะกล่าวถึงรายละเอียดของโครงสร้างข้อมูล เราควรจะมีความรู้ความเข้าใจเบื้องต้นสำหรับการศึกษาโครงสร้างข้อมูล ซึ่งได้แก่

1. การแทนที่ข้อมูลในหน่วยความจำ

ในขณะการประมวลผลด้วยคอมพิวเตอร์ ข้อมูลที่ใช้ในขณะนั้นถูกเก็บในหน่วยความจำหลัก (Memory) ซึ่งเป็นส่วนประกอบส่วนหนึ่งของคอมพิวเตอร์ เมื่อมีการใช้โครงสร้างข้อมูลจึงต้องมีการแทนที่ข้อมูลในหน่วยความจำหลักด้วย ในภาษาทางคอมพิวเตอร์ที่ใช้

ในการเขียนโปรแกรม มีวิธีการแทนที่ข้อมูลในหน่วยความจำหลักอยู่ 2 วิธีคือ (นิสาชล โตอดิเทพย์, 2535)

1. การจองเนื้อที่แบบสถิต (Static Storage Allocated) เป็นการจองที่ในหน่วยความจำด้วยขนาดที่แน่นอน โครงสร้างข้อมูลในภาษาการเขียนโปรแกรมคือ แถวลำดับ (Array) ซึ่งข้อเสียของวิธีนี้คือ ต้องกำหนดขนาดของแถวลำดับก่อนการแปล ถ้าต้องการเปลี่ยนแปลงขนาดของแถวลำดับในภายหลัง จะต้องทำการแปลโปรแกรมใหม่

2. การจองเนื้อที่แบบพลวัต (Dynamic Storage Allocated) เป็นการแทนที่ข้อมูลโดยไม่ต้องกำหนดขนาดของข้อมูลก่อนการแปล ในขณะทำงานสามารถเรียกใช้เนื้อที่หน่วยความจำเพื่อเก็บข้อมูลที่ต้องการใช้ในขณะนั้นได้ ซึ่งเป็นการประหยัดเนื้อที่หน่วยความจำ ไม่ต้องจองไว้มากจนเหลือใช้ การแทนที่ข้อมูลลักษณะนี้มีในภาษาการเขียนโปรแกรมบางภาษาเท่านั้น เช่น ภาษาปาสคาล ซี อัลกอ 68 พีเอลวัน ประเภทของข้อมูลในการขอใช้เนื้อที่หน่วยความจำแบบพลวัตนี้คือ ตัวชี้ที่อยู่ (Pointer)

2. <u>การวิเคราะห์ประสิทธิภาพของขั้นตอนวิธี (Algorithms)</u>

ขั้นตอนวิธี (Algorithms) เป็นขั้นตอนหรือวิธีการทำงานที่ชัดเจน และสามารถกระทำให้สิ้นสุดได้ในเวลาหนึ่ง ๆ ในการแก้ปัญหา ซึ่งถ้าต้องการแก้ปัญหาโดยคอมพิวเตอร์ ขั้นตอนการแก้ปัญหาจึงต้องเขียนด้วยภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่ง เช่น ภาษาซี จึงอาจกล่าวได้ว่าโปรแกรมที่เขียนขึ้นเพื่อแก้ปัญหาก็คือ ขั้นตอนวิธี (นิสาชล โตอดิเทพย์, 2535)

ในการแก้ปัญหาหนึ่ง ๆ อาจมีวิธีหรือหนทางในการแก้ปัญหานั้นมากกว่าหนึ่งวิธีการ การตัดสินใจเลือกวิธีการใดต้องพิจารณาถึงปัจจัย 2 ประการ คือ (สุชาย ธนวเสถียร และ วิชัย จิวังกูร, 2529)

1. วิธีการนั้นใช้เนื้อที่หน่วยความจำมากน้อยเพียงใด

2. มีวิธีการ ขั้นตอนการทำงานที่รวดเร็วเพียงใด

ปัจจัยแรกนั้น ขึ้นอยู่กับการเลือกโครงสร้างข้อมูลและลักษณะการแทนข้อมูลที่เหมาะสม ส่วนปัจจัยหลังนั้นขึ้นอยู่กับขั้นตอนวิธีการที่ใช้ในการโปรแกรมว่าดีเพียงใด โดยจุดประสงค์หลักแล้ว เราต้องการวิธีการที่ทำงานได้เร็วที่สุดและใช้เนื้อที่ในหน่วยความจำน้อยที่สุด แต่ในบางครั้งเราอาจไม่ได้ทั้ง 2 อย่างพร้อมกัน หรืออาจจะได้แต่ต้องใช้เวลาหรือทรัพยากรมากในการพัฒนา อาจไม่คุ้มค่าในการใช้งานหรือไม่ทันกับความต้องการใช้ นอกจากปัจจัยดังกล่าวข้างต้นแล้ว เราควรจะพิจารณาถึงขั้นตอนวิธีที่สามารถตรวจหาจุดบกพร่องได้สะดวก และทำความเข้าใจ

กับขั้นตอนวิธีนั้นได้ง่ายด้วย เนื่องจากมูลค่าของการเขียนและแก้ไขโปรแกรมอาจจะมีมากกว่ามูลค่าของความเร็วในการทำงานและหน่วยความจำที่ใช้ (ประกาศิต ชาติบุรุษ และ อาทิตย์ จิตต์จุฬานนท์, 2533)

ในการวัดประสิทธิภาพของขั้นตอนวิธีมีการวัดประสิทธิภาพในรูปของฟังก์ชันของจำนวนข้อมูล n ฟังก์ชันนั้นเรียกว่า ฟังก์ชัน บิ๊ก-โอ (big-oh) ซึ่งคิดจากจำนวนครั้งของการเปรียบเทียบในขั้นตอนวิธีนั้น ข้างล่างนี้เป็นตัวอย่างฟังก์ชัน O ที่เรียงจากน้อยไปมาก.

O (1) หมายถึงขั้นตอนวิธีนั้นใช้เวลาคงที่ไม่ขึ้นกับจำนวนข้อมูล

$O(\log_2 n)$

$O(n)$

$O(n\log_2 n)$

$O(n^2)$

$O(n^3)$

$O(2^n)$

อนึ่งสัญลักษณ์ O นั้นเป็นการบ่งบอกว่าเราพิจารณาค่าฟังก์ชันที่ n มีค่ามากๆ เราต้องดูว่า f(n) นั้นมีเทอมใดเป็นเทอมเด่น เช่น ถ้า $f(n) = 4n^2 + 5n + 6$ แล้ว จะได้ว่า O[f(n)] มีค่าเป็น $O(n^2)$ ค่าคงที่ 4 และ เทอม 5n + 6 ถูกตัดไป ทั้งนี้เพราะค่า f(n) เมื่อ n เป็นอนันต์จะคุมโดยค่าของเทอม n^2 (สุชาย ธนวเสถียร และ วิชัย จิวังกูร, 2529)

นิยามของฟังก์ชัน O

กล่าวได้ว่า g(n) = O(f(n)) ถ้ามีค่าคงที่ k และ n_0 ซึ่ง $g(n) <= k\left|f(n)\right|$ สำหรับทุกๆ $n >= n_0$ (นิสาชล โตอดิเทพย์, 2535)

ตัวอย่าง หาฟังก์ชัน O ของ 3n(n+1)/2

$3n(n+1)/2 <= 3n^2$ ทุก ๆ $n >= n_0$ เมื่อ $k = 3$, $n_0 = 1$

ดังนั้น $3n(n+1)/2 = O(n^2)$

<u>โครงสร้างข้อมูลแบบต้นไม้</u>

โครงสร้างข้อมูลแบบต้นไม้สามารถแทนที่อยู่ในหน่วยความจำได้ทั้งแบบสถิตและแบบพลวัต นอกจากนั้นยังอาจมีข้อกำหนดให้มีลักษณะพิเศษอื่น ๆ ทำให้เกิดเป็นโครงสร้างข้อมูลต้นไม้รูปแบบต่าง ๆ เช่นต้นไม้แบบทวิภาค (Binary Tree), ต้นไม้ AVL, และรวมทั้งโครงสร้างข้อมูล

แบบฮีปด้วย เราจึงควรมีความเข้าใจข้อกำหนดหลัก ๆ ของโครงสร้างข้อมูลแบบต้นไม้โดยทั่ว ๆ ไปก่อนที่เราจะศึกษาถึงโครงสร้างข้อมูลต้นไม้ในรูปแบบอื่น

ต้นไม้ในธรรมชาติจะมีจุดที่แตกกิ่งก้านสาขาออกไป และงอกจากข้างล่างขึ้นข้างบน โดยแต่ละจุดจะมีความสัมพันธ์กันในกิ่งก้านสาขานั้น โครงสร้างข้อมูลแบบต้นไม้ในระบบคอมพิวเตอร์ก็เช่นเดียวกัน โดยที่จุดแตกกิ่งก้านเราเรียกว่า บัพ (node) ซึ่งเราจะให้เป็นตัวเก็บข้อมูลหรือข่าวสารไว้ เส้นที่ต่อระหว่างบัพ จะแสดงความสัมพันธ์ระหว่างบัพ เราเรียกว่า กิ่ง (Branch) ส่วนการวาดแผนภาพต้นไม้ เราจะวาดจากบนลงล่าง ซึ่งตรงข้ามกับต้นไม้จริง (ประกาศิต ชาติบุรุษ และ อาทิตย์ จิตต์จุพานนท์, 2533)

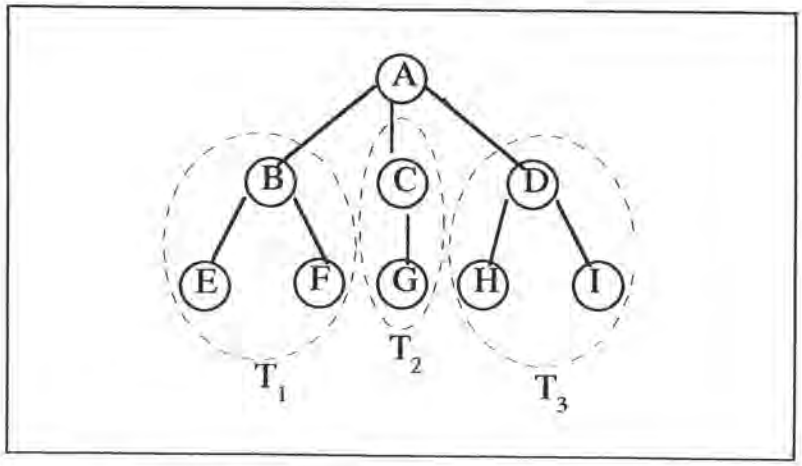
1. นิยามของโครงสร้างข้อมูลแบบต้นไม้

โครงสร้างข้อมูลแบบต้นไม้มีคำนิยามดังนี้ (นิสาชล โตอดิเทพย์, 2535)

ต้นไม้ประกอบด้วยสมาชิกที่เรียกว่า บัพ (Node) ที่

- ว่าง (ไม่มีบัพในต้นไม้) หรือ
- มีบัพหนึ่งถือเป็นบัพราก (Root Node) ส่วนบัพที่เหลือแบ่งเป็นต้นไม้ย่อย T_1, T_2,.....,T_k ($k >= 0$) โดยต้นไม้ย่อยมีคุณสมบัติเป็นต้นไม้เช่นกัน

ความสัมพันธ์ระหว่างบัพรากและบัพรากของต้นไม้ย่อยเป็นไปในลักษณะแม่กับลูก คือบัพรากเป็นบัพแม่ และบัพรากของต้นไม้ย่อยเป็นลูก



รูปที่ 2.1 โครงสร้างข้อมูลแบบต้นไม้ (นิสาชล โตอดิเทพย์, 2535)

ตัวอักษรหรือตัวเลขในบัพเป็นชื่อของบัพหรืออาจจะเป็นค่าที่อยู่ในบัพ ต้นไม้ในรูป 2.1 มีบัพ A เป็นบัพรากของต้นไม้ ต้นไม้ย่อยของ A มี 3 ต้นคือ ต้นไม้ T_1, T_2 และ T_3 ที่มี B, C และ D เป็นบัพราก ต้นไม้ B มีต้นไม้ E และ F เป็นต้นไม้ย่อย ต้นไม้ C มีต้นไม้ G เป็นต้นไม้ย่อย ต้นไม้ D มี H และ I เป็นต้นไม้ย่อย หรือพูดอีกอย่างว่า บัพ A เป็น บัพแม่ของ B, C และ D บัพ B เป็นบัพแม่ของ E และ F บัพ C เป็นบัพแม่ของ G เป็นต้น หรือในทางกลับกัน บัพ B, C และ D เป็นบัพลูกของบัพ A

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

กิ่ง (Branch) หรือ เอดจ์ (Edge) คือเส้นที่เชื่อมต่อระหว่างบัพแม่กับลูก บัพที่ไม่มีลูก เช่น บัพ E, F, G, H และ I เรียกว่า บัพใบ (Leaf Node)

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

ดีกรี (Degree) ของบัพ X ใด ๆ คือ จำนวนลูกของบัพนั้น เช่น ดีกรีของบัพใบเท่ากับ 0

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 บัพแม่ของ X นั้น เช่น บัพ E และ F มีบัพ B เป็นบัพที่มาก่อนทันที และบัพ B มี A เป็นบัพที่มาก่อนทันที

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

ระดับ (Level) เป็นหมายเลขแสดงระดับของบัพในต้นไม้ บัพรากถูกกำหนดให้อยู่ในระดับ 0 ส่วนบัพที่มาทีหลังทันทีอยู่ในระดับถัดไปคือ 1 ถ้าบัพ X (ยกเว้นบัพใบ) ใด ๆ อยู่ในระดับ i ลูกของบัพ X จะอยู่ในระดับ i + 1

ความสูงหรือความลึก (Height หรือ Depth) ของต้นไม้คือ ระดับสูงสุดของต้นไม้นั้น จากรูป 2.1 บัพ A อยู่ระดับ 0 บัพ H และ I อยู่ในระดับ 2 ดังนั้น ต้นไม้นี้มีความสูงเท่ากับ 2

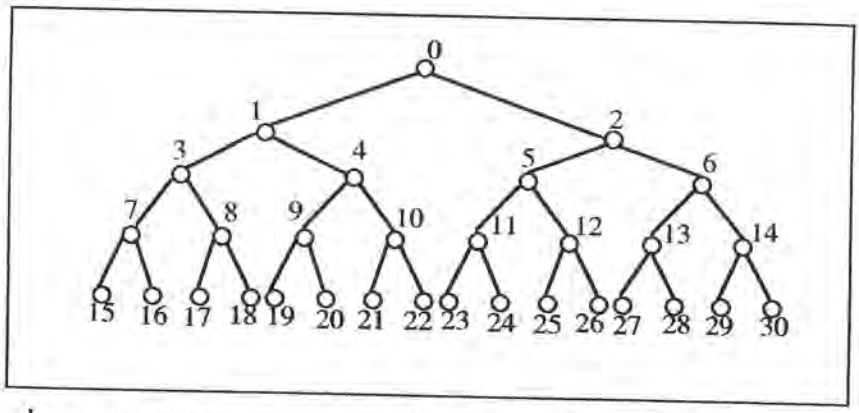
2. ต้นไม้แบบทวิภาค (Binary Tree)

เป็นต้นไม้ที่ว่างเปล่า หรือ ประกอบด้วยบัพที่เป็นบัพรากพร้อมกับต้นไม้ย่อยทางซ้ายและทางขวาของบัพรากซึ่งเป็นต้นไม้แบบทวิภาคเช่นกัน (Kruse and others, 1991)

โครงสร้างข้อมูลแบบฮีป

เมื่อเรากล่าวโครงสร้างข้อมูลแบบฮีปแล้ว ส่วนมากมักจะหมายถึงฮีปชนิดทวิภาค (Binary Heap) ดังนั้น ในการกล่าวถึงโครงสร้างข้อมูลแบบฮีปในครั้งต่อ ๆ ไป โดยไม่มีส่วนขยายอื่นแล้ว ก็จะหมายถึง โครงสร้างข้อมูลแบบฮีปชนิดทวิภาค (Weiss, 1992)

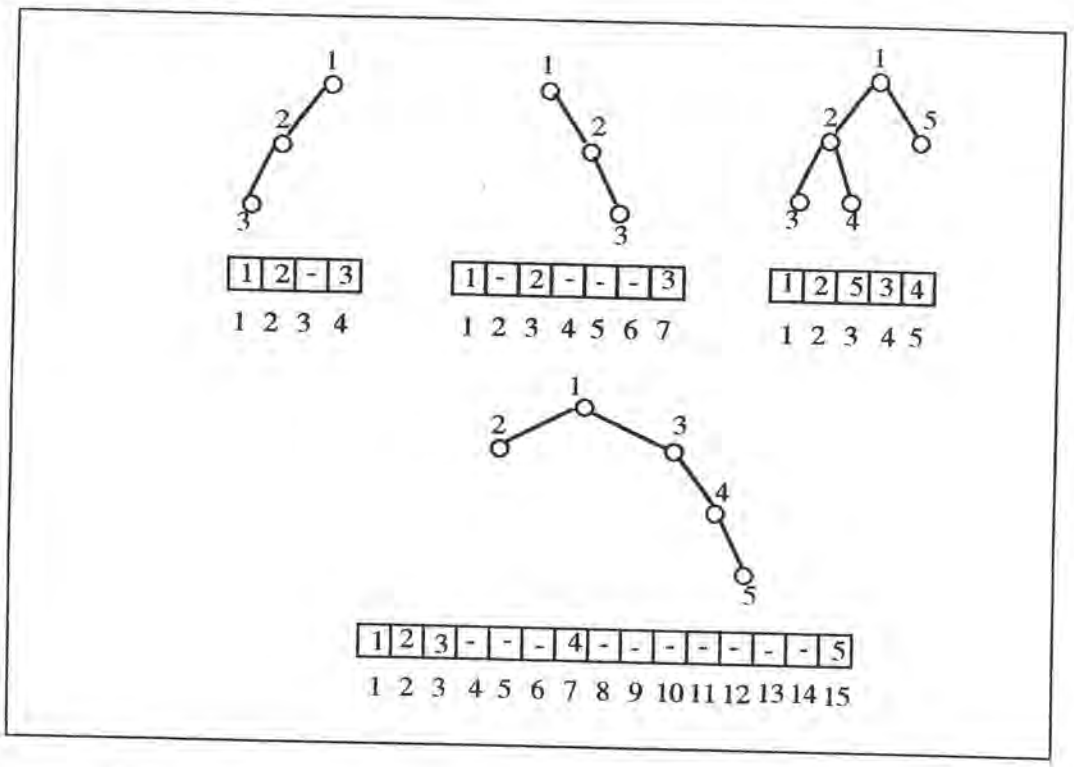
โครงสร้างข้อมูลแบบฮีปจัดเป็นต้นไม้แบบทวิภาค (Binary tree) แบบหนึ่ง ซึ่งเหมาะสมที่จะเก็บอยู่บนพื้นที่ต่อเนื่อง โดยเป็นต้นไม้แบบทวิภาคที่พยายามปรับให้เป็นต้นไม้แบบสมบูรณ์



รูปที่ 2.2 ต้นไม้ทวิภาคแบบสมบูรณ์ที่มี 31 บัพ (Kruse and others, 1991)

ต้นไม้ทวิภาคแบบสมบูรณ์ (Complete binary tree) เป็นโครงสร้างข้อมูลต้นไม้แบบทวิภาคที่สามารถเก็บอยู่ในพื้นที่แบบต่อเนื่อง (Contiguous lists) โดยแต่ละบัพ (Node) หมายถึง แต่ละหน่วยข้อมูลที่จะเก็บลงในพื้นที่แบบต่อเนื่อง ซึ่งก็คือ แถวลำดับ (Array) แต่ละช่อง จากตัวอย่างต้นไม้ทวิภาคแบบสมบูรณ์ในรูป 2.2 เราสามารถสรุปได้ว่า บัพที่ k โดย k เริ่มต้นที่ 0 จะมีลูกที่อยู่ทางซ้ายและทางขวาของบัพ ซึ่งจะจัดเก็บอยู่ในตำแหน่งที่ 2k+1 และ 2k+2 ของแถวลำดับ ถ้าค่าของ 2k+1 หรือ 2k+2 นี้อยู่เกินขนาดของแถวลำดับที่เก็บโครงสร้างข้อมูลแสดงว่าบัพที่ k นั้นไม่มีบัพที่เป็นลูกอยู่ในโครงสร้างต้นไม้นั้น

ในความเป็นจริงแล้ว ข้อมูลในตำแหน่งต่าง ๆ ที่เก็บอยู่ในแถวลำดับขนาดต่าง ๆ สามารถนำมาแสดงเป็นต้นไม้แบบทวิภาครูปแบบต่าง ๆ ได้ ดังรูป 2.3 ซึ่งจะเห็นได้ว่า ต้นไม้แบบทวิภาคที่ไม่เป็นต้นไม้แบบสมบูรณ์นั้น จะทำให้เกิดการสูญเสียพื้นที่บางส่วนไปโดยเปล่าประโยชน์เมื่อจัดเก็บข้อมูลอยู่ในพื้นที่แบบต่อเนื่อง แต่ถ้าเป็นต้นไม้แบบสมบูรณ์ จะไม่เกิดพื้นที่สูญเปล่าบางส่วนขึ้นมาเลย



รูปที่ 2.3 ต้นไม้ทวิภาคที่เก็บอยู่ในพื้นที่แบบต่อเนื่อง (Kruse and others, 1991)

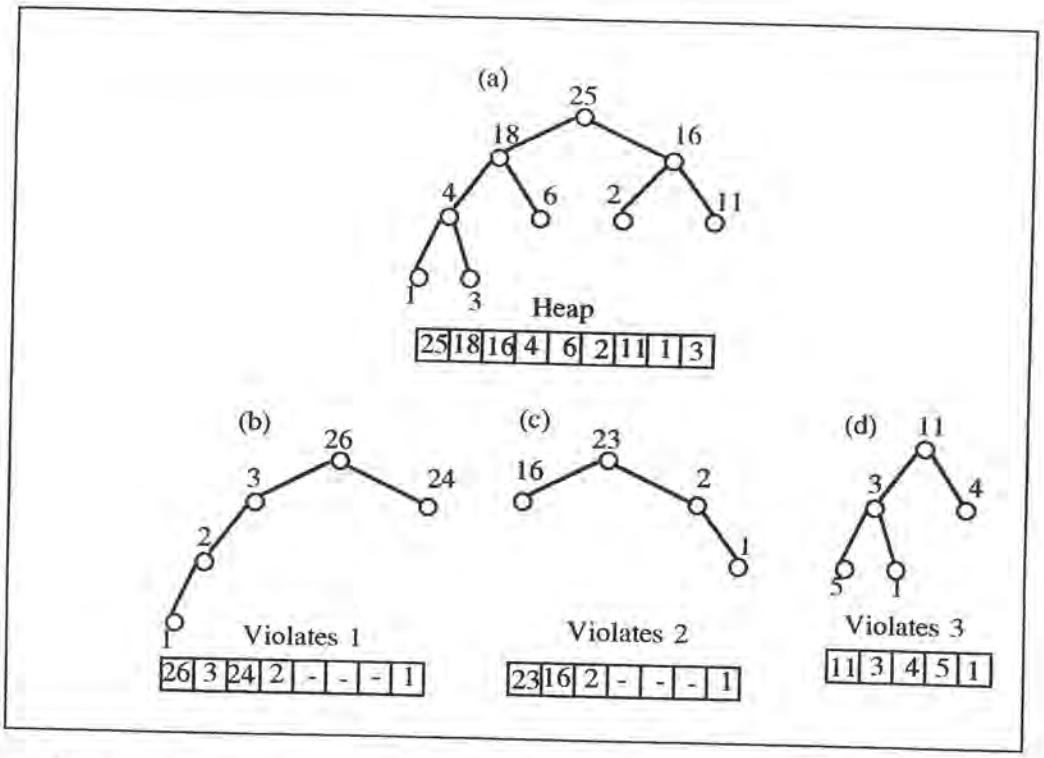
โครงสร้างข้อมูลแบบฮีปในรูปแบบของต้นไม้แบบทวิภาคประกอบด้วยบัพข้อมูลต่าง ๆ โดยคีย์ของแต่ละบัพข้อมูลมีลักษณะที่สำคัญดังนี้ (Kruse and others, 1991)

1. บัพใบทั้งหมดจะอยู่ในระดับเดียวกันหรือระดับที่ติดกัน

2. บัพใบทั้งหมดที่อยู่ระดับล่างสุด จะเกิดจากข้างซ้ายมาก่อน และบัพในระดับอื่นที่มาก่อน จะถูกใช้ไปหมดแล้ว (ยกเว้นระดับล่างสุด)

3. คีย์ของข้อมูลที่อยู่ในบัพรากนั้น จะมีค่ามากกว่าหรือเท่ากับคีย์ของข้อมูลที่อยู่ในบัพลูกหลาน (กรณีเป็นฮีปแบบค่ามากอยู่ข้างบน) และต้นไม้ย่อยด้านซ้ายและด้านขวาของบัพใด ๆ ก็มีลักษณะเป็นฮีปเช่นกัน

ข้อกำหนดใน 2 ข้อแรก เป็นการทำให้เกิดการใช้พื้นที่ต่อเนื่องในการเก็บข้อมูลให้มีประสิทธิภาพที่สุด ส่วนข้อกำหนดข้อสุดท้าย ทำเพื่อให้เกิดการจัดเรียงลำดับ โดยที่บัพรากจะเป็นคีย์ที่ใหญ่ที่สุดในโครงสร้างข้อมูลแบบฮีป



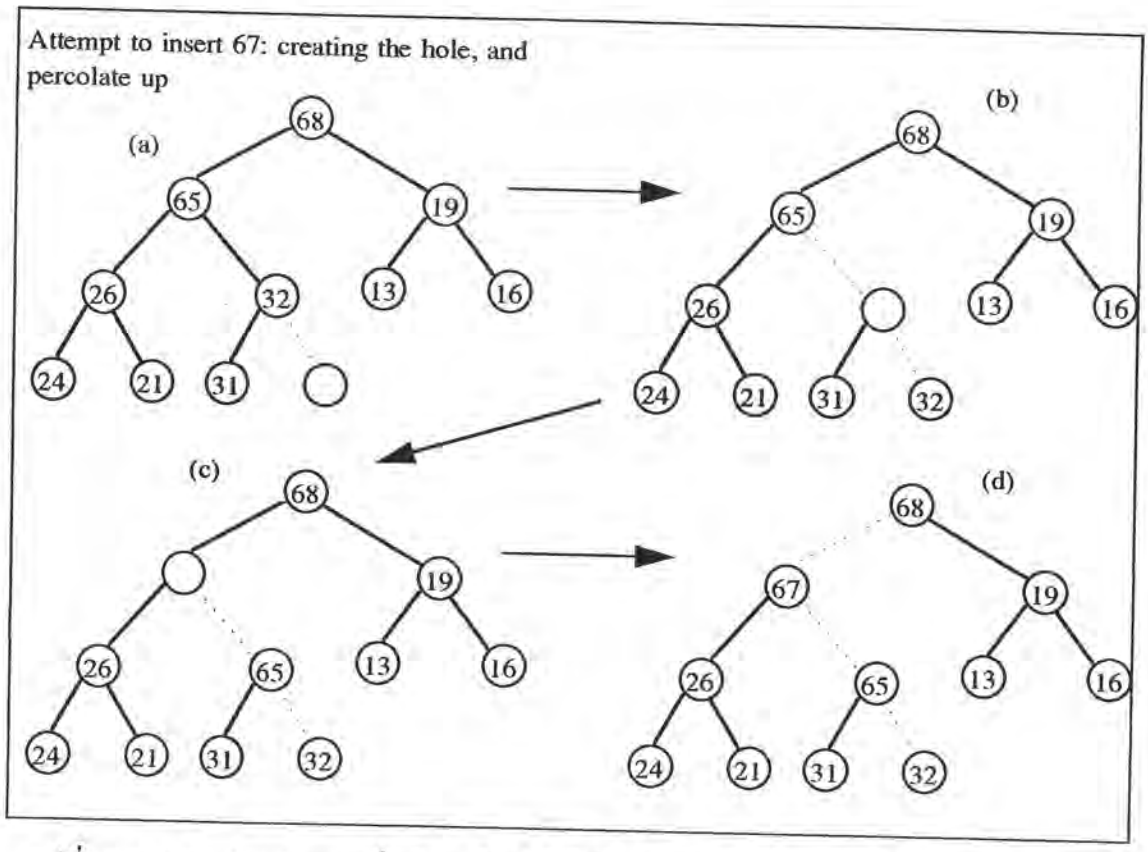
รูปที่ 2.4 โครงสร้างข้อมูลแบบฮีปและต้นไม้แบบอื่น ๆ (Kruse and others, 1991)

ในรูปที่ 2.4 นี้ รูป (a) จะแสดงถึงตัวอย่างโครงสร้างข้อมูลที่เป็นฮีป ส่วนในรูป (b) , (c) และ (d) จะเป็นตัวอย่างของโครงสร้างข้อมูลที่ผิดข้อกำหนดของโครงสร้างข้อมูลแบบฮีปข้อที่ 1, 2, และ 3 ตามลำดับ

1. การกระทำพื้นฐานบนโครงสร้างข้อมูลแบบฮีป (Basic Heap Operation)

เมื่อมีการกระทำใด ๆ บนโครงสร้างข้อมูลแบบฮีปไปแล้ว การกระทำนั้น ๆ จะต้องพยายามรักษาให้โครงสร้างข้อมูลยังคงมีคุณสมบัติเป็นโครงสร้างข้อมูลแบบฮีปอยู่เหมือนเดิม ซึ่งการกระทำพื้นฐานบนโครงสร้างข้อมูลได้แก่

1.1 การเพิ่มข้อมูล (Insertion) การเพิ่มข้อมูลใหม่เข้าไปในฮีปเราจะเตรียมบัพของข้อมูลใหม่ ลงในพื้นที่ว่างถัดจากบัพสุดท้ายของโครงสร้างข้อมูล แล้วจึงทำการตรวจสอบว่า บัพใหม่ที่เตรียมต่อท้ายนั้น ทำให้โครงสร้างข้อมูลมีคุณสมบัติเป็นฮีปอยู่คงเดิมหรือไม่ ถ้าเพิ่มเข้าไปแล้ว ทำให้ไม่เป็นฮีปก็จะทำการย้ายค่าของบัพแม่ที่อยู่ข้างบนใส่ลงไปที่บัพใหม่ที่สร้างขึ้น และจะทำการเปรียบเทียบและย้ายค่าในลักษณะเช่นนี้ไปเรื่อย ๆ ในทิศทางขึ้นไปหาบัพราก จนกว่าตำแหน่งที่บัพใหม่ไปอยู่แล้ว ทำให้โครงสร้างข้อมูลมีคุณสมบัติเป็นฮีปเหมือนเดิม ในรูปที่ 2.5 จะแสดงตัวอย่างการเพิ่มข้อมูลใหม่ที่มีค่าเป็น 67 เข้าไปในโครงสร้างข้อมูล โดยเริ่มต้นจากรูป (a) บัพค่า 67 ที่สร้างเข้าไปต่อท้ายในโครงสร้างข้อมูลนี้ ทำให้คุณสมบัติของฮีปผิดไป จึงทำการย้ายค่าของบัพ 32 โดยเลื่อนบัพ 32 ลงมาแทนที่บัพใหม่ที่เตรียมไว้ดังรูป (b) แล้วทำการเปรียบเทียบบัพค่า 67 กับบัพแม่ที่อยู่ในระดับถัดขึ้นไปอีก ซึ่งมีค่าเป็น 65 หลังจากการเปรียบเทียบแล้วจึงย้ายค่าโดยเลื่อนบัพค่า 65 ลงมาดังในรูป (c) และจะทำเช่นนี้ไปจนกระทั่งได้ตำแหน่งที่เหมาะสมสำหรับบัพใหม่ซึ่งมีค่า 67 จึงหยุด โดยได้ผลดังรูป (d) (Weiss, 1992)

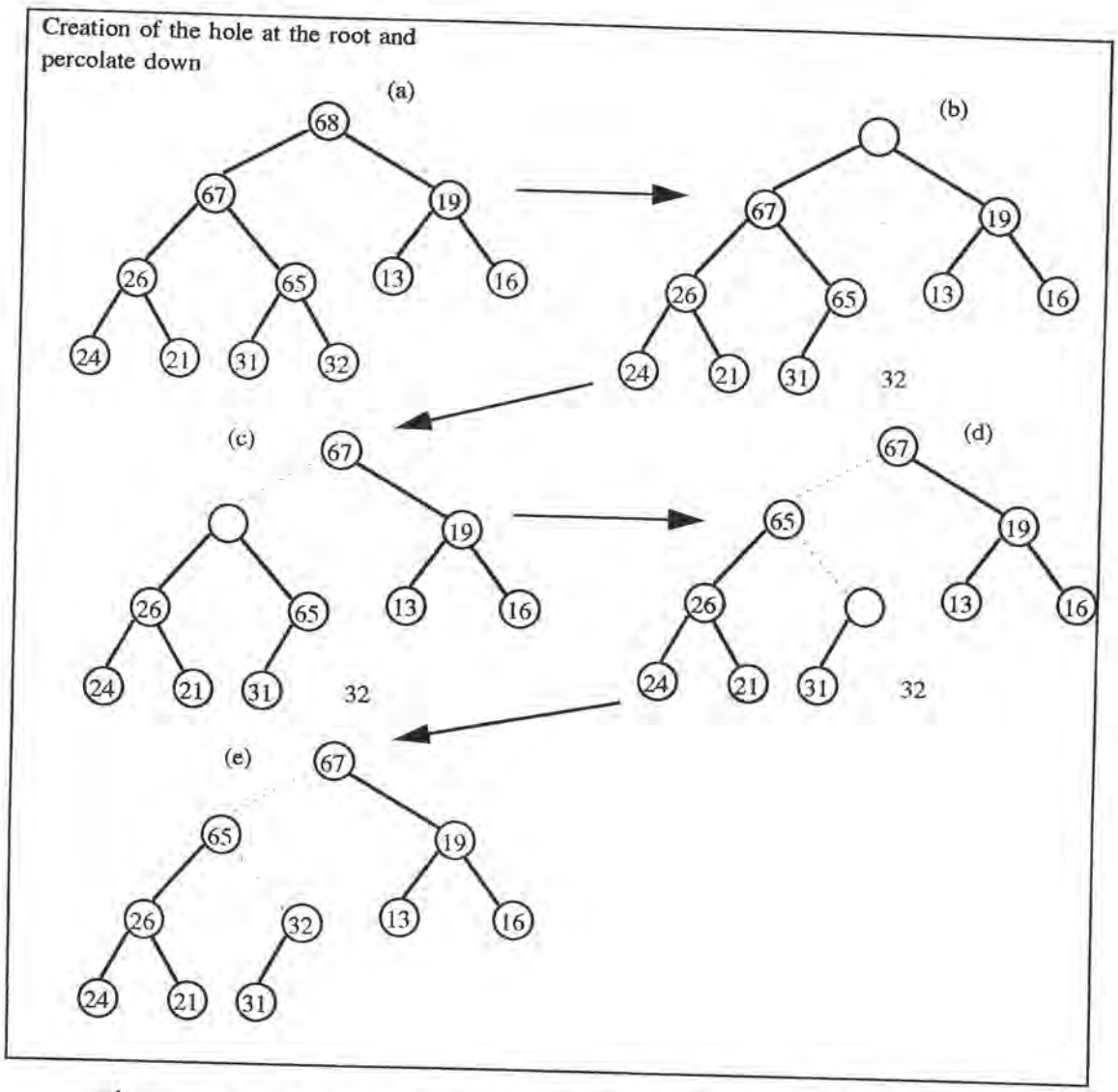


รูปที่ 2.5 แสดงตัวอย่างการเพิ่มข้อมูลใหม่เข้าไปในโครงสร้างข้อมูลแบบฮีป (Weiss, 1992)

วิธีการทำงานข้างต้นเป็นวิธีการที่เรียกว่า "การเคลื่อนย้ายขึ้นบน" (Percolate Up) บัพของข้อมูลใหม่จะถูกเคลื่อนย้ายในโครงสร้างข้อมูลในทิศทางที่ขึ้นข้างบนจนกระทั่งได้ตำแหน่งที่ถูกต้อง

1.2 การลบข้อมูล (Deletion) ในที่นี้จะกล่าวถึงในกรณีที่เป็นโครงสร้างข้อมูลฮีปชนิดที่กำหนดให้ข้อมูลที่มากที่สุดอยู่ข้างบน การลบข้อมูลนี้ส่วนมากจะหมายถึงการอ่านข้อมูลที่มากที่สุดออกมาจากฮีป การลบข้อมูลมากที่สุดออกจากโครงสร้างข้อมูลฮีปทำได้ง่ายมากก็คือ การลบข้อมูลบัพราก ซึ่งเป็นบัพแรกสุดของโครงสร้างข้อมูลฮีปเสมอ เมื่อบัพที่มากที่สุดถูกลบออกไปพร้อมกับจำนวนข้อมูลลดลงไป 1 บัพ ทำให้บัพรากเป็นบัพว่าง และข้อมูลบัพสุดท้ายในโครงสร้างข้อมูลฮีปจะต้องหาตำแหน่งที่อยู่ใหม่ จึงทำการเปรียบเทียบบัพข้อมูลสุดท้ายกับบัพข้างซ้าย ข้างขวาของบัพราก แล้วย้ายข้อมูลของบัพที่มีค่าที่มากกว่าเข้าไปในบัพราก และย้ายบัพว่างลงมาข้างล่าง 1 ระดับ เราจะทำวิธีการเหล่านี้ซ้ำไปเรื่อยจนกระทั่งได้ตำแหน่งที่ถูกต้องสำหรับบัพว่างที่จะเก็บข้อมูลบัพสุดท้ายของโครงสร้างข้อมูลฮีปและบัพรากก็ยังคงเก็บข้อมูลที่มีค่ามากที่สุดในโครงสร้างข้อมูลเช่นเดิม

ในรูปที่ 2.6 เริ่มต้นจากรูป (a) แสดงภาพจำลองของโครงสร้างข้อมูลฮีปเมื่อมีการลบข้อมูลบัพรากออกจากโครงสร้างข้อมูล หลังจากข้อมูลในบัพรากซึ่งมีค่า 68 ถูกลบออกไปแล้ว เราก็จะต้องหาตำแหน่งที่อยู่ใหม่ให้กับบัพข้อมูลค่า 32 ซึ่งเป็นบัพสุดท้ายของโครงสร้างข้อมูล ดังแสดงในรูป (b) บัพค่า 32 ไม่สามารถเข้าไปแทนที่ในบัพรากซึ่งว่างอยู่ได้ เพราะจะทำให้คุณสมบัติของฮีปเสียไป เราจึงต้องย้ายค่าของบัพลูกของบัพราก ซึ่งมีค่า 67 เพราะว่ามีค่ามากกว่าเข้าไปในบัพราก และเลื่อนบัพว่างลงมา 1 ระดับ (รูปที่ 2.6 c) เราทำซ้ำเช่นเดิมอีก โดยการย้ายค่าบัพ 65 เข้าไปในบัพว่าง และเลื่อนบัพว่างลงมาอีก 1 ระดับ (รูปที่ 2.6 d) ซึ่งบัพว่างนี้ก็จะเป็นที่สำหรับข้อมูลของบัพค่า 32 (รูปที่ 2.6 e) ขั้นตอนในการลบบัพข้อมูลมากที่สุดนี้ เป็นวิธีการที่เรียกว่า "การเคลื่อนย้ายลงล่าง" (Percolate down) (Weiss, 1992)

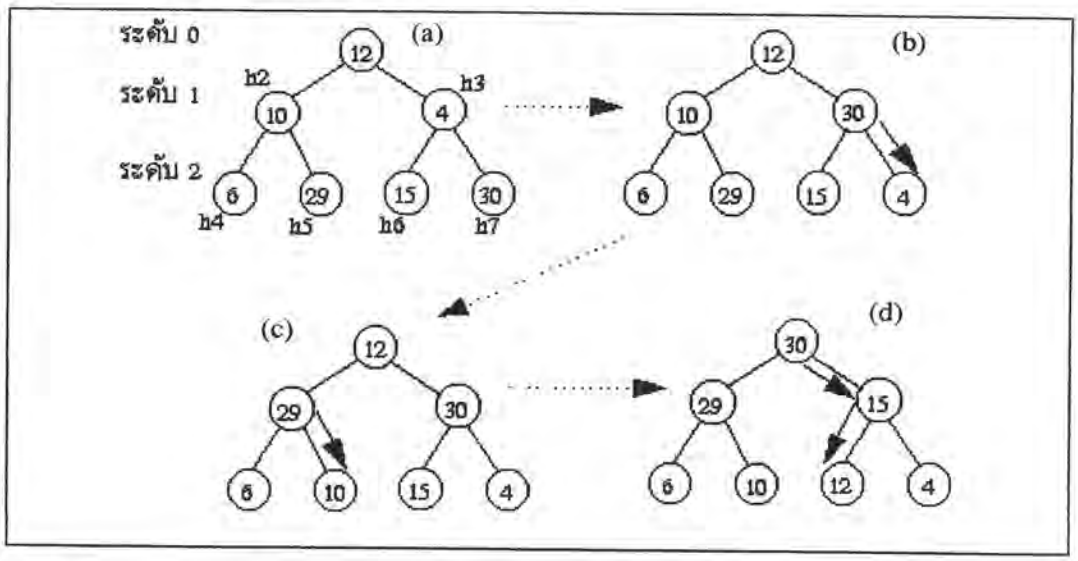


รูปที่ 2.6 แสดงตัวอย่างการลบบัพรากออกจากโครงสร้างข้อมูลแบบฮีป (Weiss, 1992)

1.3 การสร้างโครงสร้างข้อมูลฮีป (Building a Heap) เป็นการนำชุดข้อมูลของโครงสร้างข้อมูลต้นไม้ทวิภาคแบบสมบูรณ์ที่อยู่ในลำดับที่ไม่เป็นระเบียบ มาทำให้เป็นโครงสร้างข้อมูลแบบฮีป เป็นที่ทราบว่าโครงสร้างข้อมูลที่มีบัพเดียวจะเข้าคุณสมบัติของฮีปเสมอ ดังนั้น เราจึงไม่จำเป็นต้องกังวลเหล่าบัพที่เป็นใบของโครงสร้างข้อมูล นั่นคือ บัพที่เป็นครึ่งหลังของโครงสร้างข้อมูล เราจึงเริ่มพิจารณาจากบัพกึ่งกลางของโครงสร้างข้อมูลขึ้นไปถึงบัพราก เราสามารถประยุกต์ใช้วิธีการของการเคลื่อนย้ายลงล่าง (Percolate down) มาใช้ในการสร้างโครงสร้างข้อมูลฮีป

ในรูปที่ 2.7 เป็นการแสดงตัวอย่างการสร้างโครงสร้างข้อมูลฮีปจากโครงสร้างข้อมูลต้นไม้ ในรูป (a) เราเริ่มจาก h_3 ถูกแทนที่ด้วยลูกขวา h_7 (รูป 2.7 b) เนื่องจาก

h_7(30) มีค่ามากกว่าลูกซ้าย h_6(15) และมากกว่า h_3(4) และลงไปเปรียบเทียบกับ h_7 ต่อไป แต่ h_7 ไม่มีลูกแล้ว และถูกเปลี่ยนค่าเป็น h_3 บัพต่อไปคือ h_2 ถูกแทนด้วย h_5 ด้วยวิธีการเดียวกับ h_3 (รูป 2.7 c) และบัพสุดท้ายที่พิจารณาคือ h_1 ถูกแทนด้วยลูกทางขวา h_3(30) ซึ่งมีค่ามากกว่าลูกซ้าย h_2 (29) และมากกว่า h_1(12) ลงมาที่ h_3 มีค่าใหม่เป็น 12 ถูกแทนค่าด้วยลูกทางซ้าย h_6(15) ซึ่งมีค่ามากกว่าลูกทางขวา h_7 และมากกว่า h_3 แล้วลงไปพิจารณาต่อไปที่ h_6 แต่ h_6 ไม่มีลูกแล้ว ขั้นตอนการปรับต้นไม้ให้เป็นฮีปจึงเสร็จสิ้นได้ผลดังรูป 2.7 d



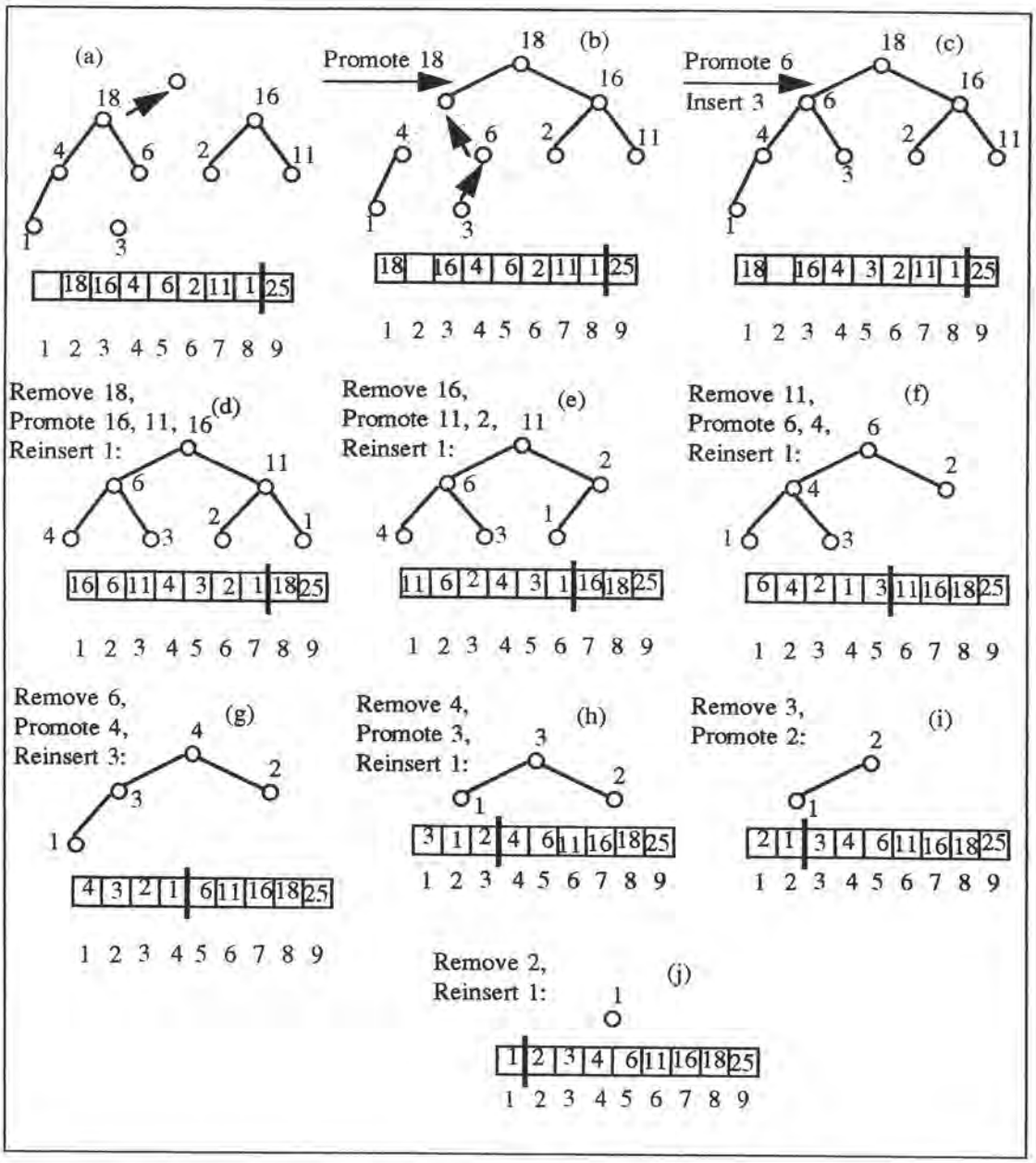
รูปที่ 2.7 แสดงตัวอย่างการสร้างโครงสร้างข้อมูลให้เป็นฮีป (นิสาชล โตอดิเทพย์, 2535)

ให้สังเกตว่า การแทนที่ค่าใน h_i (ที่ไม่ใช่บัพใบ) ต้องทำไปในเส้นทางลูกทางซ้ายหรือขวา จนถึงบัพใบ เช่น ปรับค่าใน h_1 ทำในเส้นทาง h_1 - h_3 - h_6

2. <u>การจัดเรียงลำดับแบบฮีป (Heapsort)</u>

Heapsort เป็นการจัดเรียงลำดับข้อมูลในโครงสร้างข้อมูลวิธีหนึ่ง ซึ่งทำการจัดเรียงจากโครงสร้างข้อมูลที่เป็นชนิดฮีปที่เก็บอยู่บนพื้นที่แบบต่อเนื่อง เนื่องจากโครงสร้างข้อมูลแบบฮีปมีบัพรากซึ่งเป็นหน่วยข้อมูลแรกในโครงสร้างข้อมูลมีค่าของข้อมูลมากที่สุด(สำหรับกรณีโครงสร้างข้อมูลแบบฮีปชนิดที่ค่ามากอยู่ข้างบนและต้องการเรียงลำดับจากน้อยไปมาก) เราจึงสามารถย้ายข้อมูลบัพรากเข้าไปแทนที่บัพสุดท้ายในโครงสร้างข้อมูลได้ทันที ซึ่งจะทำให้เราต้องหาที่อยู่ตำแหน่งใหม่ที่เหมาะสมในโครงสร้างข้อมูลให้บัพสุดท้ายที่เพิ่งถูกบัพรากย้ายเข้าไปอยู่ ขณะเดียวกันบัพรากที่เพิ่งย้ายข้อมูลออกไปก็ว่างลง ต้องหาบัพข้อมูลที่เหมาะสมเข้าไปแทนที่ แล้วทำการลด

จำนวนบัพข้อมูลที่จะพิจารณาในรอบต่อไปลงหนึ่ง กระทำซ้ำเช่นนี้ไปเรื่อยๆจนกระทั่งเหลือบัพข้อมูลเพียงบัพเดียวให้พิจารณาจึงหยุด เราก็จะได้โครงสร้างข้อมูลต้นไม้แบบฮีปที่ได้เรียงลำดับแล้ว ในรูปที่ 2.8 แสดงภาพจำลองการเรียงลำดับข้อมูลในโครงสร้างข้อมูลต้นไม้แบบฮีปจากโครงสร้างข้อมูลในรูปที่ 2.4 a



รูปที่ 2.8 แสดงตัวอย่างการจัดเรียงลำดับข้อมูลแบบ Heapsort (Kruse and others, 1991)

ในรูปย่อย (a), (b), และ (c) ของรูปที่ 2.8 แสดงการทำงานในรอบแรก ที่ย้ายบัพราก 25 เข้าไปในบัพ 3 แล้วจึงเปรียบเทียบระหว่างบัพลูกซ้ายขวาของบัพรากกับบัพ 3 ซึ่งเป็นบัพสุดท้ายที่เพิ่งถูกแทนที่ ก็จะได้บัพ 18 ที่มีค่ามากกว่า ย้ายขึ้นไปแทนที่ในบัพราก และกระทำเช่นนี้ลงไปอีกในต้นย่อยที่บัพรากของต้นไม้ย่อยนั้นย้ายเข้าไปแทนที่ในบัพรากของต้นไม้ใหญ่ ดังนั้นค่าที่มากที่สุดระหว่างบัพ 4 , 6 และ 3 ซึ่งก็คือบัพ 6 ย้ายขึ้นไปในบัพ 18 เดิมและลงไปพิจารณาบัพ 6 แต่บัพ 6 เป็นบัพใบที่ไม่มีลูกหลานแล้ว จึงย้ายค่าของบัพ 3 ลงไปในบัพ 6 เดิม

ในรอบแรกของการทำงานนี้ ค่าของข้อมูลที่มากที่สุด ก็จะไปอยู่ที่บัพสุดท้ายของโครงสร้างข้อมูล แล้วจึงเริ่มการทำงานในรอบต่อไป โดยที่ขนาดของโครงสร้างข้อมูลที่จะพิจารณาได้ลดลง 1 เราจะเริ่มย้ายข้อมูลบัพรากไปที่บัพสุดท้ายของโครงสร้างข้อมูลอีก ในรูปย่อยตั้งแต่รูปที่ 2.8 d ขึ้นไป จะเป็นผลของการทำงานในแต่ละรอบของการเรียงลำดับข้อมูลแบบฮีป

3. <u>การวิเคราะห์ประสิทธิภาพของ Heapsort</u>

วิธีการจัดเรียงลำดับข้อมูลแบบ Heapsort นั้นจะได้ประสิทธิภาพไม่ดีมากนักในกรณีที่โครงสร้างข้อมูลที่ต้องการจัดเรียง มีจำนวนข้อมูลน้อย แต่ในกรณีที่มีจำนวนข้อมูลมาก Heapsort เป็นวิธีการจัดเรียงลำดับข้อมูลที่มีประสิทธิภาพอีกวิธีหนึ่ง สำหรับข้อมูลจำนวน n ตัวที่มีการจัดเก็บอยู่ในพื้นที่แบบต่อเนื่อง Heapsort จะสามารถทำได้เสร็จในเวลา O(nLog(n)) เสมอ และเป็นวิธีการที่ต้องการใช้พื้นที่เพิ่มเติมน้อยที่สุด โดยเป็นจำนวนที่คงที่ (Kruse and others, 1991)

จากการวิเคราะห์แล้วพบว่า ชุดข้อมูลที่เข้ามาจัดเรียงในกรณีเฉลี่ยและกรณีเลวที่สุด (Worst case เช่น ถ้าต้องการจัดเรียงจากมากไปหาน้อย แต่ข้อมูลที่อยู่ในโครงสร้างเป็นแบบน้อยไปหามากอยู่ก่อนแล้ว) Heapsort จะใช้เวลาไม่แตกต่างกันมากนักในทั้ง 2 กรณี ถ้าลักษณะข้อมูลที่เข้ามาเป็นแบบเลวที่สุดแล้ว วิธีการของ Heapsort จะดีกว่าวิธีการจัดเรียงแบบ Quicksort และMergesort (เฉพาะ Mergesort ที่จัดเก็บข้อมูลบนพื้นที่แบบต่อเนื่อง) โดยสรุปแล้ว Heapsort เหมาะสำหรับการทำงานที่ต้องการรับประกันประสิทธิภาพ คือสามารถทำงานได้ดีสม่ำเสมอใกล้เคียงกันตลอด โดย Heapsort สามารถหลีกเลี่ยงเหตุการณ์ที่เกิดการด้อยประสิทธิภาพลงอย่างรุนแรงได้ (Kruse and others, 1991)

4. การประยุกต์ใช้โครงสร้างข้อมูลแบบฮีป

โครงสร้างข้อมูลแบบฮีปส่วนมาก จะประยุกต์ใช้ในงานจัดเรียงข้อมูล แต่ไม่เหมาะสมที่จะใช้ในงานค้นหาข้อมูล จากการประยุกต์ใช้สำหรับการจัดลำดับข้อมูล โครงสร้างข้อมูลแบบฮีปจึงอาจเรียกตามการประยุกต์ใช้งานได้ว่า "แถวคอยบุริมภาพ" (Priority Queues) โดยที่แถวคอยบุริมภาพเป็นโครงสร้างข้อมูลที่มีการทำงานเพียง 2 อย่างเท่านั้นคือ

1. การเพิ่มข้อมูล
2. การนำข้อมูลที่มีค่าสูงสุด (หรือต่ำสุด) ออกมาใช้

ตัวอย่างการประยุกต์ใช้งานของแถวคอยบุริมภาพได้แก่

- การเลือกข้อมูลที่มีค่ามากที่สุด หรือน้อยที่สุดออกมาได้ทันที (Selection Problem) เนื่องจากข้อมูลดังกล่าวเป็นข้อมูลตัวแรกสุดของแถวคอยบุริมภาพหรือบัพราก จึงสามารถอ่านข้อมูลออกมาได้ทันทีโดยไม่ต้องเสียเวลาค้นหา แต่ต้องเสียเวลาอีกส่วนหนึ่งในการจัดข้อมูลที่เหลืออยู่ในแถวคอยให้มีคุณสมบัติเหมือนเดิม (Weiss, 1992)

- ใช้ในระบบปฏิบัติการคอมพิวเตอร์ (Operating System) เช่นในระบบปฏิบัติการแบบการแบ่งการใช้เวลา (Time-Sharing) ซึ่งมีงานจำนวนมากที่รอการใช้หน่วยประมวลผลกลาง (CPU) และแต่ละงานก็อาจมีลำดับความสำคัญที่ไม่เท่ากัน ดังนั้นจึงนำงานที่รอเข้าไปทำงานในหน่วยประมวลผลกลาง มาจัดให้อยู่ในแถวคอยบุริมภาพตามลำดับความสำคัญหรือตามลำดับก่อนหลังของการเข้ามาในแถวคอยก็ได้ (Kruse and others, 1991)

- ใช้ในการจัดลำดับก่อนหลังของงานที่จะรอออกผลลัพธ์ทางเครื่องพิมพ์ โดยเงื่อนไขของการจัด อาจจัดตามลำดับของการเข้ามาในแถวคอยก่อนหรือหลัง หรืออาจใช้เงื่อนไขของการให้งานที่มีจำนวนหน้าที่น้อยได้เข้าไปพิมพ์ก่อนงานที่มีจำนวนหน้ามากเพื่อไม่ให้เกิดการรอคอยในแถวคอยนานเกินควร (Weiss, 1992)

- ใช้ในงานจำลองเหตุการณ์ (Event Simulation) เช่นการจำลองงานสนามบิน หรือเพื่อศึกษาลักษณะแถวคอยของลูกค้าที่มารอใช้บริการของธนาคาร (Kruse and others, 1991)

การเขียนโปรแกรมเชิงทัศน์ (Visual Programming)

การเขียนโปรแกรมที่มีลักษณะการเชื่อมประสานกับผู้ใช้โดยใช้รูปภาพ (Graphic User Interface) ได้เป็นที่ต้องการของผู้ใช้มากขึ้น เมื่อไมโครซอฟต์วินโดว์ ตั้งแต่รุ่น 3.0 ขึ้นไปประสบ

ความสำเร็จทางด้านการตลาด ระบบงานที่มีลักษณะดังกล่าวข้างต้น ช่วยให้ผู้ใช้งานคอมพิวเตอร์เกิดความพอใจในการใช้งานมากขึ้น แต่การพัฒนาโปรแกรมที่จะทำงานภายใต้สภาพปฏิบัติการของไมโครซอฟต์วินโดว์ สำหรับผู้เขียนโปรแกรมแล้ว มีความยุ่งยากอยู่มาก เพราะชุดตัวแปลภาษา (Compiler) ที่มีในตลาดและหาได้ง่ายนั้น มีเพียงภาษาซี และภาษาปาสคาล ซึ่งหาผู้เชี่ยวชาญอย่างแท้จริงใน 2 ภาษานี้ได้ไม่มากนัก นอกจากนี้ยังต้องเขียนโปรแกรมร่วมกับชุดเครื่องมือในการพัฒนาของไมโครซอฟต์ที่ชื่อว่า Software Development Kit (SDK) ซึ่งเป็นการเรียกใช้ฟังก์ชันต่าง ๆ ที่มีชื่อเรียกโดยทั่วๆไปว่า Windows Application Programming Interface หรือเรียกสั้นๆว่า API ฟังก์ชัน การเรียกใช้ฟังก์ชันเหล่านี้ไม่สะดวกนัก เพราะผู้ใช้จะต้องเข้าใจ และต้องจดจำวิธีการเรียกใช้ อีกทั้งยังต้องปฏิบัติตามกฎเกณฑ์ที่ได้กำหนดไว้ของแต่ละฟังก์ชัน จากการพัฒนาโปรแกรมภายใต้สภาพปฏิบัติการวินโดว์ที่มีความยุ่งยาก แต่ได้ระบบงานที่มีการติดต่อกับผู้ใช้แบบรูปภาพที่สร้างความสะดวกและความพอใจให้แก่ผู้ใช้มากขึ้น ทำให้มีการพัฒนาแนวคิดในการเขียนโปรแกรมรูปแบบใหม่ที่เรียกว่าการเขียนโปรแกรมเชิงทัศน์ (Visual Programming) ซึ่งมีวิธีในการเขียนโปรแกรม ที่สะดวกมากขึ้นกว่าเดิม

การเขียนโปรแกรมเชิงทัศน์เป็นสภาพแวดล้อมหรือวิธีการที่ผู้เขียนโปรแกรมสามารถเห็นภาพความเป็นไปขณะพัฒนาโปรแกรม เช่นเห็นภาพตัวแทนของวัตถุที่นำเข้ามาประกอบกันในโปรแกรม และวัตถุนั้นก็สามารถมีกระบวนการทำงานภายในของมันเองได้ หรืออาจสามารถเห็นภาพผลลัพธ์ของการทำงานของโปรแกรม เช่นแสดงเป็นรูปภาพกราฟิก (Graphics Image) เห็นความเปลี่ยนแปลงเป็นลำดับขั้นของการประมวลผล (Animation Sequence) (Chang, 1990)

การเขียนโปรแกรมเชิงทัศน์ ได้มีการนำแนวคิดบางส่วนของวิธีการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) เข้ามาใช้ด้วย กล่าวคือ มีวิธีการพัฒนาโปรแกรมในลักษณะแบ่งการพัฒนาออกเป็นส่วนๆ โดยแต่ละส่วนก็เป็นซอฟต์แวร์ส่วนประกอบ (Software Component) หรือก็คือวัตถุ (Object) ตามแนวคิดทางด้านการเขียนโปรแกรมเชิงวัตถุก็ได้ แล้วนำส่วนประกอบหรือวัตถุต่างๆมาประกอบเข้าด้วยกัน วิธีดังกล่าวข้างต้นจะเป็นวิธีการพัฒนาซอฟต์แวร์ที่รวดเร็วขึ้น เพราะเราสามารถนำส่วนประกอบต่างๆที่มีอยู่แล้วไปใช้ใหม่ได้ โดยไม่ต้องเสียเวลาพัฒนาส่วนประกอบที่เหมือนกันซ้ำซ้อนหลายครั้ง ในการพัฒนาซอฟต์แวร์ครั้งต่อ ๆ ไป

การที่จะพัฒนาซอฟต์แวร์ในลักษณะนำส่วนประกอบต่าง ๆ มาใช้อย่างมีประสิทธิภาพนั้นส่วนประกอบเหล่านั้นจะต้องมีวิธีการติดต่อใช้งานที่เป็นมาตรฐาน และที่สำคัญจะต้องมีซอฟต์แวร์ที่เป็นโครงสร้างหลักที่จะนำส่วนประกอบต่าง ๆ เข้ามาประกอบกันนั้น มีการติดต่อและการใช้งานที่เป็นมาตรฐานด้วยเช่นกัน นับตั้งแต่ไมโครซอฟต์วินโดว์ 3.X ได้กลายมาเป็นที่

นิยมแพร่หลายอย่างมากในโลกของการใช้งานคอมพิวเตอร์ส่วนบุคคล ลักษณะการติดต่อใช้งานของไมโครซอฟต์วินโดว์ 3.X ได้ถูกยอมรับให้เป็นมาตรฐานสำหรับซอฟต์แวร์โครงสร้างหลักในการนำซอฟต์แวร์ส่วนประกอบต่างๆเข้ามาประกอบเข้าด้วยกัน ซอฟต์แวร์ส่วนประกอบดังกล่าวเป็นส่วนที่เรียกว่าตัวควบคุม (Control) ในไมโครซอฟต์วินโดว์ (Cilwa and Duntemann, 1994)

แนวคิดเกี่ยวกับตัวควบคุม (Control) เป็นสิ่งที่มีอยู่แล้วในการเขียนโปรแกรมสำหรับวินโดว์ 3.X แต่เริ่มมีการใช้อย่างชัดเจนและเป็นมาตรฐานมากขึ้น เมื่อไมโครซอฟต์ได้ออกผลิตภัณฑ์สำหรับการเขียนโปรแกรมเชิงทัศน์ที่ชื่อว่า วิชวลเบสิค (Visual Basic) ในปี 1991 วิชวลเบสิคใช้ตัวควบคุมต่างๆ ในฐานะส่วนประกอบของการพัฒนาโปรแกรม นำตัวควบคุมมาจัดให้อยู่ในฟอร์ม (Form) ที่จะใช้งาน แล้วเขียนโปรแกรมภาษาเบสิค (BASIC) เชื่อมโยงตัวควบคุมต่าง ๆ เข้าด้วยกัน โดยที่ผู้พัฒนาโปรแกรมสามารถเห็นรูปภาพที่เป็นตัวแทนของตัวควบคุมต่าง ๆ ที่ใช้ขณะกำลังพัฒนาโปรแกรม ตลอดจนเห็นภาพผลลัพธ์ขณะกำลังประมวลผลส่งผ่านออกมาทางตัวควบคุมต่างๆเช่นกัน จากการที่วิชวลเบสิคได้ใช้แนวคิดในการนำตัวควบคุมเข้ามาประกอบกันเป็นแนวทางในการพัฒนาโปรแกรมนั้น ได้มีส่วนในการกระตุ้นและสนับสนุนให้มีการพัฒนาตัวควบคุม หรือซอฟต์แวร์ส่วนประกอบ สำหรับการใช้งานในด้านต่าง ๆ ออกมาสู่ตลาดเป็นจำนวนมาก

ตัวควบคุม (Control)

ตัวควบคุมมีแนวคิดที่คล้ายกับวัตถุ (Object) ของวิธีการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) กล่าวคือเป็นซอฟต์แวร์ส่วนประกอบ (Software Component) ที่ได้ทำไว้สำเร็จแล้ว มีวิธีการติดต่อใช้งานที่แน่นอน ประกอบด้วยองค์ประกอบ 2 ส่วนหลักคือ ข้อมูล หรือก็คือคุณสมบัติ (Property) ของตัวควบคุม และกระบวนการทำงานหรือวิธี (Method) โดยการทำงานของตัวควบคุมส่วนมากจะเป็นการทำงานที่ตอบสนองตามข้อความ (Messages) ที่ส่งมาถึงตัวควบคุม และตามสถานะของข้อมูลที่อยู่ในตารางคุณสมบัติของตัวควบคุมเอง

การใช้งานตัวควบคุมที่แท้จริงแล้วเป็นการใช้งานที่รูปเสมือน (Instance) ของตัวควบคุม โดยรูปเสมือนต่าง ๆ นั้นเกิดจากเบ้าหลอมอันเดียวกัน นั่นคือ ชั้นของตัวควบคุม (Control Class) โดยชั้นของตัวควบคุมจะเป็นต้นแบบให้กับรูปเสมือนที่สร้างขึ้นมาขณะใช้งาน ซึ่งแต่ละรูปเสมือนจะมีข้อมูลคุณสมบัติของตัวเอง ทำงานอิสระจากกันในแต่ละรูปเสมือน วิธีการนี้เองที่ทำให้เราสามารถใช้งานตัวควบคุมตัวเดียวกันในหลาย ๆ แห่งได้พร้อม ๆ กัน

ตัวควบคุมที่ใช้ในสภาพปฏิบัติการวินโดว์นั้น ส่วนมากจะเป็นแฟ้มประเภท .VBX ซึ่งก็คือตัวแทนของฟังก์ชันประเภทที่มีการเชื่อมโยงแบบพลวัตของวินโดว์ (Windows Dynamic Link Library หรือ DLL) ที่ได้ออกแบบมาเป็นพิเศษ มีกระบวนการทำงานภายในที่คล้ายกับโปรแกรมที่ทำงานภายใต้สภาพปฏิบัติการวินโดว์ ที่มีการทำงานหลักอยู่ที่การรับ ส่ง และตอบสนองต่อข้อความของตัวควบคุม และที่เด่นชัดก็คือ มีวิธีการติดต่อใช้งาน ที่ผู้ใช้สามารถเห็นภาพที่เป็นตัวแทนของตัวควบคุมได้

การนำตัวควบคุมมาเป็นซอฟต์แวร์ส่วนประกอบในการพัฒนาระบบงานนั้น มีข้อดีดังนี้คือ

1. ช่วยลดระยะเวลาและต้นทุนในการพัฒนาระบบงาน เพราะสามารถนำตัวควบคุมที่มีอยู่แล้ว กลับมาใช้ได้อีก

2. ทำให้ได้ระบบที่มีโครงสร้างเป็นระเบียบ บำรุงรักษาได้ง่าย เพราะสามารถหาจุดที่ต้องการปรับปรุงเปลี่ยนแปลงในระบบงานได้ และการเปลี่ยนแปลงไม่ทำให้เกิดผลกระทบไปยังภายนอกตัวควบคุม (Wiener and Pinson, 1992)

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

DLL ถือเป็นหลักสำคัญในการทำงานร่วมกันภายใต้สภาพปฏิบัติการวินโดว์ ชุดคำสั่ง ข้อมูล และ ทรัพยากรต่างๆ โดยมากมักจะเก็บอยู่ในแฟ้ม DLL ชุดฟังก์ชันต่างๆของ DLL ทำงานในลักษณะถูกสั่งให้ทำงานภายใต้สภาพแวดล้อมของโปรแกรมผู้เรียกใช้

แฟ้ม DLL เป็นลักษณะอย่างหนึ่งของแฟ้มกระทำการ (Executable File) ซึ่งอาจมีประเภทของแฟ้มที่แตกต่างกันเพื่อเป็นการบ่งบอกถึงหน้าที่ในการจัดการทรัพยากรต่าง ๆ ของระบบ เช่นแฟ้มประเภท .FON, .DLL, .DRV หรือ .SYS เป็นต้น โดยแฟ้มประเภท .FON เป็นทรัพยากรรูปแบบตัวอักษร แฟ้มประเภท .DRV เป็นตัวควบคุมอุปกรณ์ (Device Driver) แฟ้มประเภท .SYS เป็นแฟ้มระบบของวินโดว์ ที่มักจะทำหน้าที่ในการควบคุมอุปกรณ์บางอย่างด้วย และมี DLL บางอย่างที่ใช้ประเภทแฟ้มเป็น .EXE ด้วยเหมือนกัน DLL สามารถที่จะมีชุดคำสั่ง ข้อมูล และทรัพยากรเช่นเดียวกับโปรแกรมประยุกต์สำหรับวินโดว์ทั่วไป (Rector, 1993)

DLL นั้นไม่ได้ทำงานโดยการสั่งงานโดยตรงจากผู้ใช้งานระบบ แต่ผู้ใช้ระบบงานสั่งงานผ่านโปรแกรมประยุกต์ จากนั้นโปรแกรมประยุกต์จะเป็นผู้ทำการเรียกใช้ฟังก์ชั่นใน DLL เรียกค้นข้อมูลจาก DLL และใช้ทรัพยากรของ DLL ขณะที่ DLL ทำงานอยู่นั้นตัว DLL เองก็สามารถเรียกใช้ฟังก์ชั่น เรียกค้นข้อมูล และใช้ทรัพยากรของ DLL อื่นๆได้ด้วย

DLL ยังสามารถเรียกใช้ฟังก์ชันของโปรแกรมประยุกต์ได้ โดยฟังก์ชันนั้นจะต้องมีการกำหนดให้เป็นฟังก์ชันแบบเรียกกลับได้ (CALL BACK) ตัวอย่างของการใช้งานในรูปแบบนี้เช่น การที่ผู้ใช้ระบบเรียกใช้โปรแกรมประยุกต์ที่ทำงานภายใต้สภาพปฏิบัติการวินโดว์ ซึ่งโปรแกรมประยุกต์นั้นได้กำหนดให้เป็นฟังก์ชันแบบเรียกกลับได้อยู่ก่อนแล้ว การที่ผู้ใช้ระบบได้ทำการเรียกใช้งานโปรแกรมประยุกต์ โดยการสั่งผ่านวินโดว์ ซึ่งแท้ที่จริงแล้วก็คือการที่ DLL หลักของวินโดว์เป็นผู้เรียกใช้โปรแกรมประยุกต์นั้นมาใช้งานนั่นเอง โดย DLL หลักของวินโดว์ดังกล่าวอาจเป็น GDI, KERNEL หรือ USER ก็ได้

DLL สามารถถูกโปรแกรมประยุกต์เรียกขึ้นมาใช้งานได้ทั้งโดยทางตรงและทางอ้อม โดย DLL จะมีทรัพยากรต่าง ๆ ที่ทั้งวินโดว์และโปรแกรมประยุกต์ต้องการใช้งาน และต้องมีการใช้งานร่วมกันด้วย ซึ่ง DLL แบบหนึ่ง ๆ ภายในระบบของวินโดว์นั้น จะมีเพียงสำเนาเดียวเท่านั้นในหน่วยความจำ ซึ่งต่างกับโปรแกรมประยุกต์ของวินโดว์ซึ่งมีได้หลายสำเนาพร้อม ๆ กัน ตัวอย่างของฟังก์ชันของ DLL เช่นฟังก์ชัน GetMessage ( ) ที่ใช้ในการวนรับข้อความ (Message Loop) นั้นอยู่ในแฟ้ม DLL ชื่อ USER.EXE (Rector, 1993)

ในการพัฒนาโปรแกรมกระทำการ (.EXE Program) เราคุ้นเคยกับการเชื่อมโยงแบบสถิต (Static Linking) ซึ่งเป็นวิธีการแก้ปัญหาของการเรียกใช้ฟังก์ชันอื่นที่ไม่มีในโปรแกรม ตัวเชื่อมโยง (Linker) จะค้นหาฟังก์ชันจากคลังชุดคำสั่ง (Library) ต่าง ๆ ว่ามีหรือไม่ ถ้ามีก็จะเชื่อมโยงสำเนาของฟังก์ชันนั้นเข้ากับแฟ้มกระทำการ (.EXE File) ซึ่งทำให้ภายในแฟ้มกระทำการมีสำเนาของฟังก์ชันนั้นอยู่ด้วย ตัวอย่างเช่น การใช้ฟังก์ชัน strcpy ( ) ในโปรแกรมต่างๆ ตัวเชื่อมโยงจะรวมสำเนาของฟังก์ชัน strcpy ( ) เข้าไปในแฟ้มกระทำการ เมื่อเราสั่งในโปรแกรมเหล่านี้ให้ทำงานพร้อม ๆ กัน ก็จะมีสำเนาหลาย ๆ ชุดของฟังก์ชัน strcpy ( ) อยู่ในหน่วยความจำ (Rector, 1993)

สำหรับ DLL แล้วจะมีวิธีการเชื่อมโยงและการใช้งานของโปรแกรมขณะดำเนินงานที่แตกต่างกัน กล่าวคือ เมื่อมีการเชื่อมโยง ตัวเชื่อมโยงจะค้นหาฟังก์ชันจากข้อมูลของระเบียนนำเข้า (Input Record) ซึ่งเก็บชื่อของแฟ้ม DLL ที่มีฟังก์ชันนั้นอยู่ และหมายเลขลำดับของฟังก์ชันนั้นใน DLL หรือชื่อสัญญลักษณ์ที่เป็นตัวแทนของฟังก์ชันนั้น ตัวเชื่อมโยงก็จะเชื่อมโยงแบบพลวัต โดยการทำสำเนาของข้อมูลในระเบียนนำเข้านั้นเข้ากับแฟ้มกระทำการของโปรแกรมประยุกต์ เมื่อผู้ใช้ระบบมีการสั่งดำเนินการโปรแกรมประยุกต์ วินโดว์จะตรวจสอบว่ามี DLL อะไรที่ต้องเรียกใช้บ้าง โดยวินโดว์จะค้นหาฟังก์ชัน DLL ที่ต้องการด้วยข้อมูลจากระเบียนนำเข้านี้ และหากยังไม่มีฟังก์ชันนั้นอยู่ในหน่วยความจำ ก็ต้องมีการนำฟังก์ชันนั้นเข้าสู่หน่วยความจำก่อน แล้วจากนั้นจึงทำการเชื่อมโยงเข้ากับโปรแกรมประยุกต์ ซึ่งต่างจากวิธีการเชื่อมโยงแบบสถิตที่จะทำการเชื่อมโยง

เพียงครั้งเดียวตอนที่สร้างแฟ้มกระทำการ (.EXE File) ส่วนการเชื่อมโยงแบบพลวัตจะทำการเชื่อมโยงทุกครั้งที่มีการสั่งดำเนินงานโปรแกรม ตัวอย่างเช่นเมื่อมีการเรียกใช้ฟังก์ชันที่ชื่อว่า GetMessage ( ) ข้อมูลนำเข้าในระเบียนจะเป็นตัวแสดงให้รู้ว่าฟังก์ชันนี้มีหมายเลขประจำฟังก์ชันเป็นลำดับที่ 108 ในแฟ้ม DLL ที่มีชื่อว่า USER.EXE เมื่อมีการใช้งานจริงจากโปรแกรมประยุกต์ต่าง ๆ พร้อมกัน วินโดว์ก็จะทำการค้นหาฟังก์ชัน GetMessage ( ) จากข้อมูลในระเบียนนำเข้านั้น และหากยังไม่มีฟังก์ชัน GetMessage ( ) อยู่ในหน่วยความจำ ก็จะนำฟังก์ชัน GetMessage ( ) ไปไว้ในหน่วยความจำเพียงสำเนาเดียว แล้วเชื่อมโยงฟังก์ชัน GetMessage ( ) เข้ากับโปรแกรมประยุกต์ต่างๆ ที่เรียกใช้ฟังก์ชันนี้ (Rector, 1993)

การนำ DLL ไปใช้ในการพัฒนาโปรแกรม มีประโยชน์ดังนี้

1. ประหยัดพื้นที่ใช้งานในดิสก์ของโปรแกรมประยุกต์มากขึ้น

2. ประหยัดพื้นที่ใช้งานในหน่วยความจำของโปรแกรมประยุกต์มากขึ้น

3. การปรับปรุงแก้ไขโปรแกรมประยุกต์โดยเฉพาะส่วนที่ใช้ DLL จะกระทำได้สะดวกและรวดเร็วยิ่งขึ้น เพราะเพียงแก้ไขที่ DLL เพียงที่เดียว โดยไม่จำเป็นต้องเข้าไปแก้ไขในส่วนของโปรแกรมประยุกต์ทั้งหมด