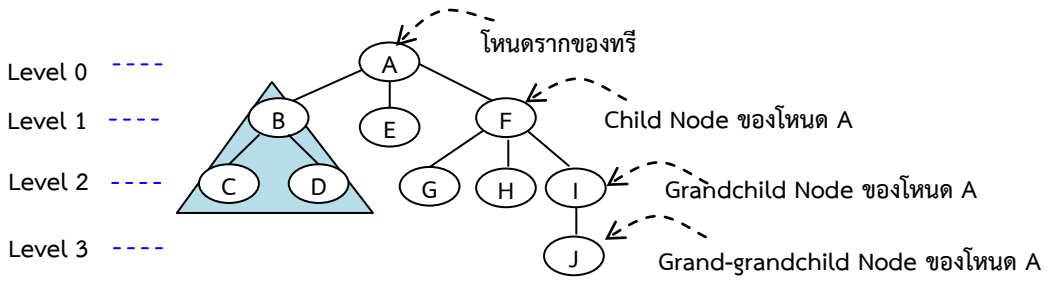
บทที่ 10

ทรี

โครงสร้างข้อมูลต้นไม้ (Tree Data Structure) หรือเรียกสั้นๆว่าทรี (Tree) เป็นโครงสร้างข้อมูลรูปแบบหนึ่งในลักษณะโครงสร้างข้อมูลชนิดไม่เชิงเส้น (Non-Linear) ที่สามารถนำไปใช้ในการจัดการกับข้อมูลได้อย่างมีประสิทธิภาพ สมาชิกแต่ละตัวในทรีสามารถเชื่อมโยงไปยังสมาชิกตัวถัดไป (Successor) ได้มากกว่าหนึ่งตัวและเชื่อมโยงถึงกันในลักษณะเป็นระดับคล้ายกับการแตกกิ่งก้านสาขาออกไปของต้นไม้ ดังนั้นความสัมพันธ์ของสมาชิกข้อมูลในทรี จึงมีลักษณะลำดับชั้น (Hierarchical Relationship) คือ มีการเชื่อมโยงของแต่ละโหนดเป็นแบบทางเดียวจากบนลงล่าง

โครงสร้าง ข้อมูลทรีประกอบด้วยโหนด (Node) สำหรับจัดเก็บข้อมูลและกิ่งหรือเส้นที่เชื่อมโยงโหนดเข้าด้วยกันเรียกว่าบรานช์ (Branch) โหนดต่างๆในทรีจะอยู่ในระดับที่ต่างกันโดยเริ่มต้นจากโหนดแรกสุดของทรี เรียกว่า โหนดราก (Root) จะอยู่ในระดับแรกสุด (Level 0) โหนดลูกๆ (Children) ของโหนดราก จะอยู่ในระดับที่ 1 (Level 1) ส่วนโหนดลูกๆ ของโหนดในระดับที่ 1 จะอยู่ในระดับที่ 2 (Level 2) ทุกโหนดในทรีจะมีความสัมพันธ์กันในลักษณะเหมือนครอบครัวเดียวกัน นั่นคือ แต่ละโหนดจะเชื่อมโยงไปยังโหนดพ่อ (Parent) ซึ่งอยู่ระดับที่เหนือกว่าได้ 1 โหนด แต่โหนดพ่อจะเชื่อมไปยังโหนดลูก (Child) ที่อยู่ระดับต่ำกว่าได้หลายโหนด โหนดลูกตั้งแต่ 2 โหนด (Children) ขึ้นไปที่เชื่อมโยงไปยังโหนดพ่อเดียวกัน จะมีความสัมพันธ์ในลักษณะเป็นโหนดพี่น้องกัน (Siblings) ดังรูปที่ 10.1 เป็นตัวอย่างโครงสร้างข้อมูลทรีที่มีโหนด A เป็นโหนดรากโหนด B E และ F เป็นโหนดพี่น้องกันเพราะต่างก็เป็นโหนดลูกของโหนด A เมื่อเพิ่มระดับของทรีไปเรื่อยๆ ก็จะมีโหนดลูกหลานเพิ่มขึ้น ดังนั้น โหนด F I และ J เป็นโหนดลูกหลาน (Descendants) ของโหนด A โดยมีสถานะเป็นโหนดลูก (Child Node) โหนดหลาน (Grandchild Node) และโหนดเหลน (Grand-grandchild Node) ของโหนด A ตามลำดับ หรือเรียกว่ารวมกันว่าโหนดลูกหลาน (Descendants) ของโหนด A และในทางกลับกันโหนด I F และ A เป็นโหนดบรรพบุรุษ (Ancestors) ของโหนด J โดยมีสถานะเป็นโหนดพ่อ (Parent) โหนดปู่ย่า-ตายาย (Grandparent) และโหนดทวด (Grand-grandparent) ของโหนด J ตามลำดับ



รูปที่ 10.1 ตัวอย่างโครงสร้างข้อมูลทรี

จากรูปที่ 10.1 สามารถอธิบายลักษณะและคุณสมบัติพื้นฐานของโครงสร้างไบนารีทรีได้ดังนี้

1) โหนด A เป็นโหนดรากของทรี
2) ทุกโหนด (ยกเว้นโหนดราก) จะมีสมาชิกตัวก่อนหน้า (Predecessor)เพียง 1 โหนด ดังนั้น โหนด B มี โหนด Aเป็น Predecessor เป็นต้น แต่สมาชิกตัวถัดไป (Successor) อาจมี 0 หรือ 1 หรือมากกว่า 1 โหนดก็ได้ดังนั้นโหนด B มี โหนด E F และ Gเป็น Successor
3) โหนดพ่อเป็นโหนดที่มี Successor ดังนั้นโหนด B คือ โหนดพ่อของโหนด E F และ G
4) โหนดลูกเป็นโหนดที่มี Predecessor ดังนั้นโหนด B คือ โหนดลูกของโหนด A
5) {B,E,F}, คือ กลุ่มโหนดพี่น้องกันที่มีโหนด A เป็นโหนดพ่อ เช่นเดียวกับ {C,D} และ {G,H,I} เป็นกลุ่มโหนดพี่น้องที่มี B และ F เป็นโหนดพ่อตามลำดับ
6) โหนดกิ่ง (Branch Node) หรือโหนดภายใน (Internal Node) คือ โหนดใดๆ ที่มีเส้นเชื่อมไปยังโหนดลูก ได้แก่ A B F และ I
7) โหนดภายนอก (External Node) หรือ โหนดใบ (Leaves) คือ โหนดใดๆ ที่ไม่มีเส้นเชื่อมไปยังโหนดลูก ได้แก่ C D E G Hและ J
8) จำนวนบรานช์ของโหนด เรียกว่า ดีกรี (Degree) เช่น
   - ดีกรีทั้งหมดของทรี คือ 9
   - ดีกรีของโหนด F คือ 3
   - ดีกรีของโหนดใบ (Leaf Node) จะเท่ากับ 0 ได้แก่ โหนด C,D,E,G,H และ J
9) เส้นทาง (Path) จากโหนด 1 ถึงโหนด k คือ ลำดับของโหนด n_1, n_2,..., n_k เมื่อ n_i คือ โหนดพ่อของโหนด n_{i+1} และ $1 \le i < k$
10) ความลึก (Depth) ของโหนด คือ ความยาวของ Path จากโหนดรากถึงโหนดนั้นเช่น
    - ความลึกของโหนดAเท่ากับ 0
    - ความลึกของโหนด J เท่ากับ 3
11) ความสูง (Height)ของโหนด คือ ความยาวของ Path จากโหนดถึงโหนดใบ
    - ความสูงของโหนด B เท่ากับ 1 และความสูงของโหนด C เท่ากับ 0
    - ความสูงของโหนด A เท่ากับ 3 ซึ่งโหนด A คือ โหนดราก จึงเท่ากับความสูงของทรี
12) ทรีว่าง (Empty tree) คือ ทรีที่ไม่มีสมาชิก หรือจำนวนโหนดในทรีเท่ากับ 0

การดำเนินการพื้นฐานกับชนิดข้อมูลนามธรรมทรี (ADT Tree Operations) ประกอบด้วย

การดำเนินการ หน้าที่

size()	นับและคืนค่าจำนวนโหนดที่จัดเก็บในทรี
isEmpty()	ตรวจสอบว่าทรีว่างหรือไม่ โดยจะคืนค่าจริงเมื่อทรีว่าง และคืนค่าเท็จเมื่อมีข้อมูลจัดเก็บในทรี
p.getItem()	ค้นคืนค่าข้อมูลที่จัดเก็บในโหนด p เมื่อ p คือ สมาชิกของทรี
p.swapItem(q)	สลับค่าข้อมูลระหว่างโหนด p และ q เมื่อ p และ q คือ สมาชิกของทรี
p.replaceItem(item)	แทนค่าข้อมูลเดิมที่จัดเก็บในโหนด p ด้วยข้อมูลใหม่ (item)
p.isRoot()	ตรวจสอบว่าโหนด p เป็นโหนดรากของทรีหรือไม่
p.isExternal()	ตรวจสอบว่าโหนด p มีสถานะเป็นโหนดภายนอกหรือไม่

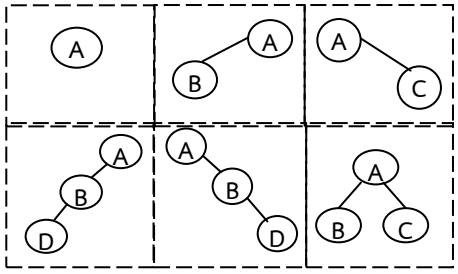
การดำเนินการ หน้าที่

p.parent()	ค้นคืนค่าโหนดพ่อของโหนด p (ควรตรวจสอบข้อยกเว้นในการเกิดข้อผิดพลาดกรณี p เป็นโหนดราก)
p.children()	ค้นหาและคืนค่ารายการโหนดลูกของโหนด p (ควรตรวจสอบข้อยกเว้นในการเกิดข้อผิดพลาดกรณี p เป็นโหนดภายนอก)
root()	ค้นหาและคืนค่าโหนดรากของทรี (ควรตรวจสอบข้อผิดพลาดในกรณีทรีว่าง)

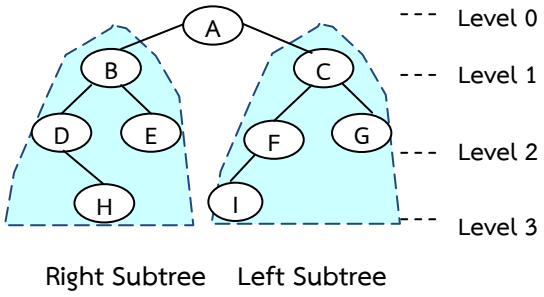
10.1 นิยามไบนารีทรี

โครงสร้างต้นไม้แบบมีทรีย่อยไม่เกิน 2 เรียกว่า ไบนารีทรี (Binary Tree) มีลักษณะเฉพาะดังนี้

1) แต่ละโหนด จะมีกิ่งหรือเส้นเชื่อมโยงไปยังโหนดลูกได้ไม่เกิน 2 โหนดเท่านั้น เรียกว่า โหนดลูกทางซ้าย (Left Child Node) และโหนดลูกทางขวา (Right Child Node) ไบนารีทรีอาจมีลักษณะเป็นทรีว่าง หรือเป็นทรีที่มี 1 2 3 หรือ n โหนดเช่นเดียวกับทรีทั่วไป ดังรูปที่ 10.2 (ก) แต่ถ้าโหนดใดมีเส้นเชื่อมไปยังโหนดลูกเพียงโหนดเดียว จะต้องระบุว่าเป็นโหนดทางซ้ายหรือโหนดทางขวา
2) กลุ่มโหนดทั้งหมดที่เชื่อมจากโหนดหนึ่งๆ เรียกว่า ต้นไม้ย่อย (Subtree) แบ่งเป็นต้นไม้ย่อยทางซ้าย (Left Subtree) และต้นไม้ย่อยทางขวา (Right Subtree) ดังรูปที่ 10.2 (ข)



(ก) ไบนารีทรีที่มี 1 2 และ 3 โหนด



(ข) Right Subtree และ Left Subtree

รูปที่ 10.2 ตัวอย่างโครงสร้างข้อมูลไบนารีทรี

จากรูปที่ 10.2 (ข) ในระดับ (Level) ที่ i ของไบนารีทรี จะมีจำนวนโหนดทั้งหมดไม่เกิน 2^i โหนด ($\leq 2^i$) เช่น ในระดับที่ 3 จะมีจำนวนโหนดไม่เกิน $2^3 = 8$ โหนด ดังนั้นหากไบนารีทรีมีความสูงเท่ากับ h จะพบว่า

- มีจำนวนโหนดที่เป็นโหนดภายนอก (External Nodes) ทั้งหมดไม่เกิน 2^h โหนด
- ไบนารีทรีแบบสมบูรณ์ที่มีจำนวนโหนดในระดับ 0 ถึงระดับที่ h ครบทั้งหมด จะมีจำนวนโหนดทั้งหมดในทรี (n) ดังสมการ

$$n = 1 + 2 + 4 + \ldots + 2^{h-1} + 2^h = 2^{h+1} - 1$$

- ค่าความสูงของทรีจะคำนวณได้จากสมการ $h = \log_2(n)$ และจำนวนโหนดที่เป็นโหนดภายนอกจะมีทั้งหมดเท่ากับ $(n + 1)/2$

การดำเนินการกับชนิดข้อมูลนามธรรมแบบไบนารีทรี (ADT Binary Tree Operations) เช่น

การดำเนินการ	**หน้าที่**
p.left()	ค้นคืนค่าโหนดลูกทางซ้ายของโหนด p(ควรตรวจสอบข้อยกเว้นในการเกิดข้อผิดพลาดกรณี p เป็นโหนดภายนอก)
p.right()	ค้นคืนค่าโหนดลูกทางขวาของโหนด p
p.parent()	ค้นคืนค่าโหนดพ่อของโหนด p (ควรตรวจสอบข้อยกเว้นในการเกิดข้อผิดพลาดกรณี p เป็นโหนดราก)
p.hasLeft()	ตรวจสอบว่าโหนด p มีโหนดลูกทางซ้ายหรือไม่ หากมีให้คืนค่าจริง หากไม่มีให้คืนค่าเท็จ
p.hasRight()	ตรวจสอบว่าโหนด p มีโหนดลูกทางขวาหรือไม่
p.isRoot()	ตรวจสอบว่าโหนด p เป็นโหนดรากหรือไม่
p.isExternal()	ตรวจสอบว่าโหนด p มีสถานะเป็นโหนดภายนอกหรือไม่
size()	นับและคืนค่าจำนวนโหนดในทรี
isEmpty()	ตรวจสอบว่าทรีว่างหรือไม่
root()	ค้นหาและคืนค่าโหนดรากของทรี (ควรตรวจสอบข้อยกเว้นในกรณีทรีว่าง)

การนำไบนารีทรีไปใช้งานจะขึ้นอยู่กับลักษณะงานที่นำไปใช้ ซึ่งมีผลให้ขั้นตอนในการทำงานของแต่ละการดำเนินการกับข้อมูลที่จัดเก็บในไบนารีทรีมีความแตกต่างกันไป ในหัวข้อนี้จะกล่าวถึงโครงสร้างข้อมูลไบนารีทรีเพื่อการค้นหาข้อมูลใน และโครงสร้างข้อมูลเอวีแอลทรี (AVL Tree)

10.2 ไบนารีทรีเพื่อการค้นหา

ไบนารีทรีเพื่อการค้นหา (Binary Search Tree) เป็นโครงสร้าง ข้อมูลไบนารีทรีที่ เก็บข้อมูลแบบมีค่าคีย์ของข้อมูลเพื่อใช้ในการค้นหาข้อมูลได้อย่างมีประสิทธิภาพ ดังนี้

1) ทุกค่าคีย์ในทรีย่อยทางซ้าย < ค่าคีย์ของโหนดราก < ทุกค่าคีย์ในทรีย่อยทางขวา ดังนั้นค่าคีย์ทั้งหมดของโหนดลูกหลานในทรีย่อยทางซ้ายของโหนดใดๆ ต้องมีค่าน้อยกว่าค่าคีย์ของโหนดนั้น และค่าคีย์ทั้งหมดของโหนดลูกหลานในทรีย่อยทางขวาของโหนดใดๆ ต้องมีค่ามากกว่าค่าคีย์ของโหนดนั้นเช่นกัน ดังรูปที่ 10. 3 (ก) หากโหนดมีค่าคีย์เท่ากับ y แสดงว่าโหนดลูกทางซ้ายจะเก็บค่าคีย์ x ใดๆ ที่มีค่าน้อยกว่า y และโหนดลูกทางขวาจะเก็บค่าคีย์ z ใดๆ ที่มีค่ามากกว่าค่า y จากรูปที่ 10.3 (ข) พบว่าโหนดของค่าคีย์ 30 จะมีค่าคีย์ของโหนดลูกหลานในทรีย่อยทางซ้ายน้อยกว่า 30 ทั้งหมด คือ 15 20 25
2) จำนวนโหนดสูงสุดของทรีที่มีความสูง h จะมีค่าเท่ากับ $2^{(h+1)}$-1 เช่น h เท่ากับ 2 จำนวนโหนดสูงสุด จะเท่ากับ 2^3-1 = 7 โหนด

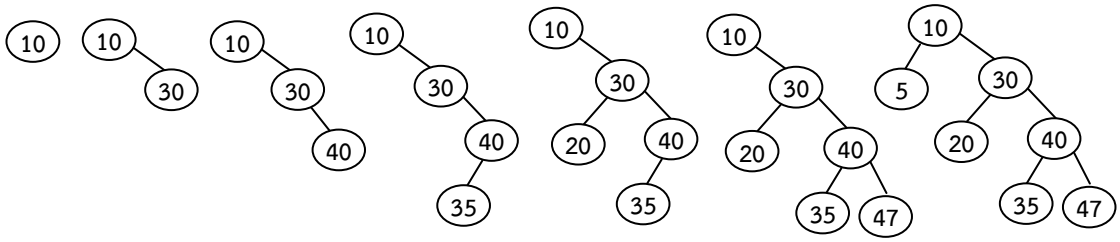
(ก) คุณสมบัติของไบนารีทรีเพื่อการค้นหา (ข) ตัวอย่างไบนารีทรีเพื่อการค้นหา

รูปที่ 10.3 คุณสมบัติและตัวอย่างโครงสร้างข้อมูลไบนารีทรีเพื่อการค้นหา

(1) การเพิ่มโหนดใหม่ในไบนารีทรีเพื่อการค้นหา

การดำเนินการเพิ่มโหนด ข้อมูลใหม่ (Insert Operation) ในโครงสร้างไบนารีทรีเพื่อการค้นหา กรณีทรีว่าง จะกำหนดโหนดใหม่เป็นโหนดราก แต่หากไม่ใช่ทรีว่างจะแทรกโหนดใหม่ ดังนี้

1) กำหนดให้โหนดปัจจุบัน (เรียกว่าโหนด x) ชี้ที่โหนดราก
2) เปรียบเทียบค่าคีย์ของโหนดใหม่แต่ละโหนดกับโหนด x
3) หากค่าคีย์ของโหนดใหม่ มีค่ามากกว่าค่าคีย์ของโหนด x แยกเป็น 2 กรณีดังนี้
   - กรณี x ไม่มีโหนดลูกทางขวา (x เป็นโหนดใบ) แสดงว่าพบตำแหน่งในการแทรกโหนดใหม่ จะแทรกโหนดใหม่ไว้ทางขวาของโหนด x
   - กรณี x มีโหนดลูกทางขวา จะกำหนดให้โหนด x ชี้ที่โหนดลูกทางขวาแล้ววนกลับไปทำข้อ 2 ซ้ำเพื่อหาตำแหน่งที่ต้องการแทรกโหนดใหม่ต่อไป
4) หากค่าคีย์ของโหนดใหม่ มีค่าน้อยกว่าค่าคีย์ของโหนด x แยกเป็น 2 กรณีดังนี้
   - กรณี x ไม่มีโหนดลูกทางซ้าย (x เป็นโหนดใบ) แสดงว่าพบตำแหน่งในการแทรกโหนดใหม่ จะแทรกโหนดใหม่ไว้ทางซ้ายของโหนด x
   - กรณี x มีโหนดลูกทางซ้าย จะกำหนดให้โหนด x ชี้ที่โหนดลูกทางซ้ายแล้ววนกลับไปทำข้อ 2 ซ้ำเพื่อหาตำแหน่งที่ต้องการแทรกโหนดใหม่ต่อไป
5) วนทำซ้ำข้อ 1 จนกว่าจะแทรกข้อมูลใหม่จนครบข้อมูลนำเข้าทั้งหมดดังรูปที่ 10.4 แสดงตัวอย่างการแทรกโหนดใหม่ของข้อมูล10 30 40 35 20 47 และ 5 ตามลำดับ

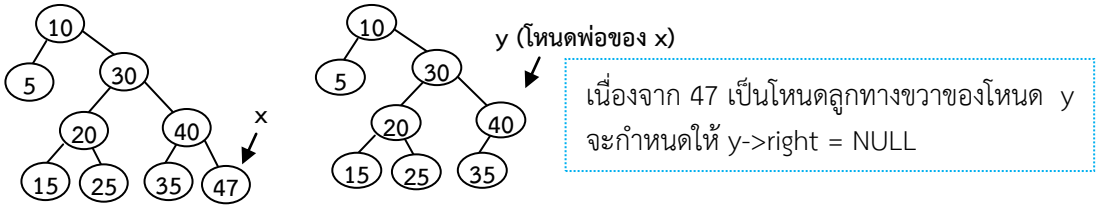


รูปที่ 10.4 การเพิ่มข้อมูลในไบนารีเพื่อการค้นหาด้วยข้อมูล 10 30 40 35 20 42 และ 5 ตามลำดับ

(2) การลบโหนดในไบนารีเพื่อการค้นหา

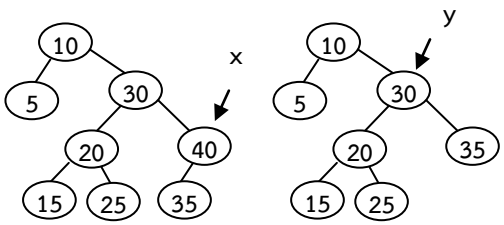
การดำเนินการลบโหนด (Delete Operation) ในทรี จะตัดการเชื่อมโยงโหนดออกจากโครงสร้างไบนารีเพื่อการค้นหา กำหนดให้ x ชี้ที่โหนดที่ต้องการลบออกจากทรี วิธีการลบขึ้นอยู่กับสถานะของโหนด x ที่ต้องการจะลบแยกเป็น 3 กรณี ดังนี้

1) กรณี x คือ โหนดใบ ให้กำหนดโหนดพ่อของ x ชี้ไปยังค่า null ส่วนจะเป็นการกำหนดให้ตัวชี้ทางซ้ายหรือตัวชี้ทางขวาเป็นค่า null จะขึ้นอยู่กับสถานะของโหนด x ว่าเป็นโหนดลูกทางซ้ายหรือโหนดลูกทางขวา ดังรูปที่ 10.5 แสดงตัวอย่างการลบโหนด 47 ซึ่งเป็นโหนดใบ



รูปที่ 10.5 กรณีโหนดที่ต้องการลบเป็นโหนดใบ (โหนด 47)

2) กรณี x มีโหนดลูกเพียงโหนดเดียว จะให้กำหนดให้ตัวชี้ของโหนดพ่อของ x ชี้ไปยังโหนดลูกของ x ดังรูปที่ 10.6 แสดงตัวอย่างการลบโหนด 40 ที่มีลูกทางซ้ายเพียงโหนดเดียว



เนื่องจาก 40 เป็นโหนดลูกทางขวาของโหนด y (ค่า 30) และโหนด 40 มีลูกทางซ้ายอยู่หนึ่งโหนด (ค่า 35) เมื่อลบโหนด 40 จะกำหนดให้ y->right = x->left เพื่อให้ 35 เป็นลูกทางขวาของ 30 แทน

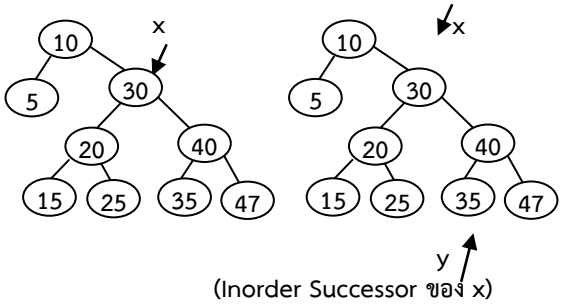
รูปที่ 10.6 กรณีโหนดที่ต้องการลบ (โหนด 40) มีโหนดลูกเพียงโหนดเดียว (โหนด 42)

1) กรณีโหนด x มีทั้งโหนดลูกทางซ้ายและทางขวา

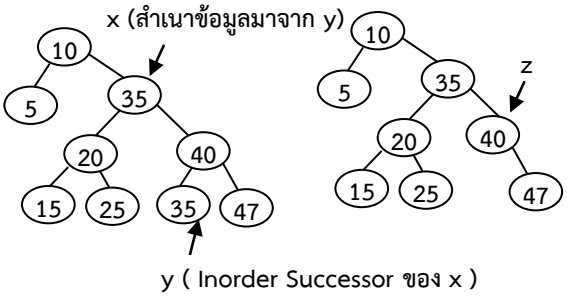
3.1) จากโหนดทั้งหมดในทรีย่อยทางขวาของโหนด x (x' s Right Subtree) ให้หาตำแหน่งโหนด y ที่มีค่าคีย์น้อยที่สุด (Leftmost Node) เรียกว่าโหนดตามหลัง (Successor) แบบอินออร์เดอร์ของโหนด x ในกรณีการเข้าถึงข้อมูลในไบนารีทรีแบบอินออร์เดอร์ เนื่องจากการเข้าถึงข้อมูลในไบนารีทรีเพื่อการค้นหาเป็นแบบเรียงลำดับจากน้อยไปมาก ซึ่งเป็นลักษณะการเข้าถึงแบบอินออร์เดอร์นั่นเอง

3.2) สำเนา (Copy) ข้อมูลจากโหนด y ไปไว้ที่โหนด x แล้วลบโหนด y แบ่งเป็น 2 กรณีดังนี้

- กรณีโหนด y คือ โหนดใบ ให้กำหนดโหนดพ่อของ y ชี้ไปยังค่า null ในลักษณะเดียวกันกับข้อ 1 ดังตัวอย่างในรูปที่ 10.7
- กรณีโหนด y มีโหนดลูก 1 โหนด ให้กำหนดโหนดพ่อของ y ชี้ไปยังโหนดลูกของ y ในลักษณะเดียวกันกับข้อ 2 ดังตัวอย่างในรูปที่ 10.8



(ก) หาตำแหน่งโหนด y จากทรีย่อยทางขวาของ x



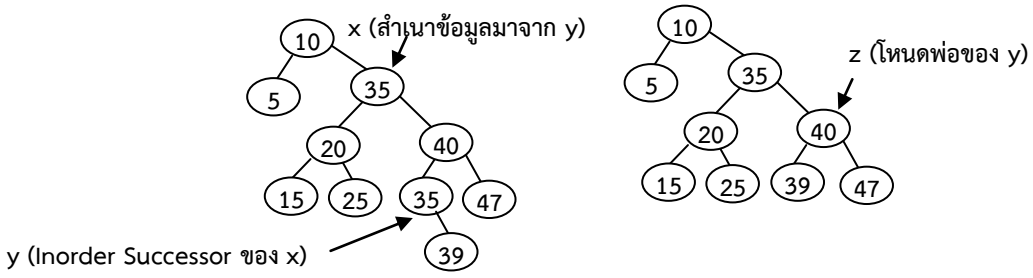
(ข) สำเนาจาก y ไป x แล้วลบโหนด y

เมื่อจะลบโหนด 30 ซึ่งมีทั้งโหนดลูกทางซ้ายและโหนดลูกทางขวา มีการทำงานดังนี้

1) หาโหนด y ที่เป็น Inorder Successor ของ 30 ซึ่งก็คือค่าน้อยสุดของทรีย่อยทางขวาของโหนด 30 ในกรณีนี้จะพบว่า y คือ ค่า 35 ดังรูปที่ 10.7 (ก)
2) สำเนาข้อมูลจากโหนด 35 ไปไว้ที่โหนด 30
3) ลบโหนด 35 ออกจากทรี
   แยกเป็น 2 กรณี ดังนี้
   - กรณี 35 ไม่มีโหนดลูก จะกำหนดให้ z->left = NULL 35 ดังรูปที่ 10.7 (ข)
   - กรณี 35 มีโหนดลูกทางขวา จะกำหนดให้ z->left = y->right ดังรูปที่ 10.8

(เนื่องจากโหนด yคือ ลูกทางซ้ายของโหนด z (ค่า 40))

รูปที่ 10.7 กรณีโหนดที่ต้องการลบ (โหนด x) มีโหนดลูก 2 โหนด และโหนด y ไม่มีลูก



รูปที่ 10.8 กรณีโหนดที่ต้องการลบ (โหนด x) มีโหนดลูก 2 โหนด และโหนด y มีโหนดลูกทางขวา

(3) การค้นหาคีย์ในไบนารีเพื่อการค้นหา

การดำเนินการเพื่อการค้นหา (Find Operation) เป็นการค้นหาตำแหน่งโหนดที่มีค่าคีย์ตรงกับค่าคีย์ที่ต้องการค้นหา ตัวอย่างเช่น หากต้องการลบโหนดที่มีค่าคีย์เท่ากับ k จะต้องทำการค้นหาตำแหน่งโหนดที่ต้องการลบด้วยการเรียกใช้การดำเนินการค้นหา โดยจะเริ่มต้นที่โหนดราก แล้วเปรียบเทียบค่าคีย์ที่ต้องการค้นหากับโหนดปัจจุบัน หากพบคีย์ที่ต้องการจะหยุดค้นหา แต่หากยังไม่พบจะค้นหาไปยังโหนดถัดไป จนกว่าจะพบโหนดที่มีค่าคีย์ตรงกับคีย์ที่ต้องการค้นหา หรือจนกว่าจะถึงโหนดใบ ซึ่งจะกล่าวโดยละเอียดในหัวข้อถัดไป

(4) ประสิทธิภาพในการทำงานกับโครงสร้างไบนารีทรีเพื่อการค้นหา

กำหนดให้ n คือ จำนวนข้อมูลที่จัดเก็บในไบนารีทรี เมื่อวิเคราะห์หาประสิทธิภาพในการทำงานของการดำเนินการต่างๆ กับข้อมูลในไบนารีทรีเพื่อการค้นหา จะพบว่า

- การจัดเก็บข้อมูลในไบนารีทรีจะใช้เนื้อที่ (Space) เท่ากับ O(n)
- เวลาที่ใช้ในการทำงาน T(n) ของแต่ละการดำเนินการของไบนารีทรี ไม่ว่าจะเป็น find(),insert() หรือ remove() จะมีค่าเท่ากับความสูง (h) ของทรี คือ O(h) แบ่งเป็น 2 ลักษณะดังนี้
  - หากไบนารีทรีมีลักษณะเป็นทรีแบบสมดุล (Balance Tree) ดังรูปที่ 10.9 (ก) นั่นคือ โหนดรากมีจำนวนโหนดในทรีย่อยทางซ้ายเท่ากับจำนวนโหนดในทรีย่อยทางขวา จะพบว่าความสูงของทรีจะเท่ากับ log(n) ดังนั้น T(n) = O(log (n))
  - อาจพบกรณีที่แย่ที่สุด (Worst case) เมื่อการจัดเก็บข้อมูลในไบนารีทรีเป็นลักษณะเอียงซ้ายหรือขวาด้านเดียว ดังรูปที่ 10.9 (ข) พบว่าความสูงของทรี คือ n ดังนั้น T(n) = O(n)

จากการวิเคราะห์ประสิทธิภาพด้านเวลาในการทำงานกับข้อมูลในไบนารีทรี จะพบว่าความสมดุลของทรีเป็นเรื่องสำคัญ จึงทำให้มีผู้คิดค้นโครงสร้างข้อมูลทรีที่สามารถจัดเก็บข้อมูลในทรีแบบสมดุล เรียกว่า เอวีแอลทรี (AVL Tree) ซึ่งตั้งชื่อตามผู้คิดค้นโครงสร้างเอวีแอลทรี คือ Georgy Adelson-Velsky และ Evgenii Landis

(ก) ทรีแบบสมดุล **(ข) ทรีแบบเอียงซ้ายและเอียงขวาด้านเดียว**

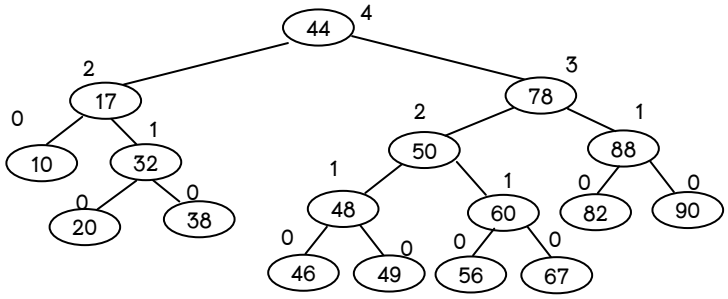
รูปที่ 10.9 โครงสร้างข้อมูลทรีแบบสมดุลและทรีแบบเอียงข้างเดียว

10.3 เอวีแอลทรี

เอวีแอลทรี (AVL Tree) เป็นไบนารีทรีเพื่อการค้นหาที่มีวิธีการดำเนินการกับทรีให้สามารถจัดเก็บข้อมูลในลักษณะสมดุลหรือเกือบสมดุล ดังรูปที่ 10.10 และโครงสร้างเอวีแอลทรีจะมีคุณสมบัติที่สำคัญ คือ

“ทุกโหนดที่ไม่ใช่โหนดใบ ทรีย่อยทางซ้ายและทรีย่อยทางขวาจะมีความสูงต่างกันมากที่สุดเพียง 1”

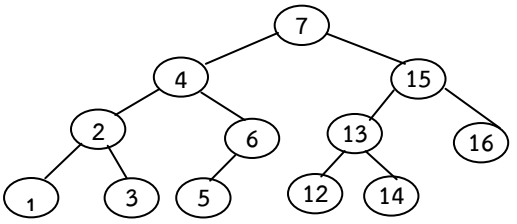
รูปที่ 10.10 แสดงตัวอย่างโครงสร้างเอวีแอลทรีที่มีความสูงเท่ากับ 4 จะพบว่าโหนด 7 8 มีความสูงของทรีย่อยทางซ้าย (มีโหนด 50 เป็นโหนดรากของทรีย่อย) เท่ากับ 2 ในขณะที่ความสูงของทรีย่อยทางขวา (มีโหนด 80 เป็นโหนดรากของทรีย่อย) เท่ากับ 1



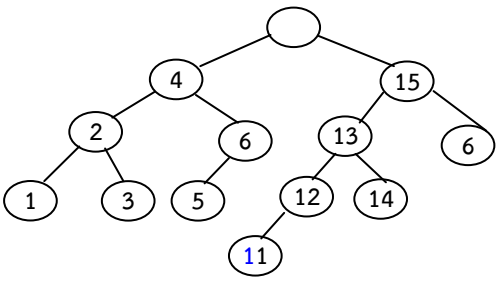
รูปที่ 10.10 ตัวอย่างโครงสร้างเอวีแอลทรี

10.3.1 การแทรกโหนดใหม่ในเอวีแอลทรี

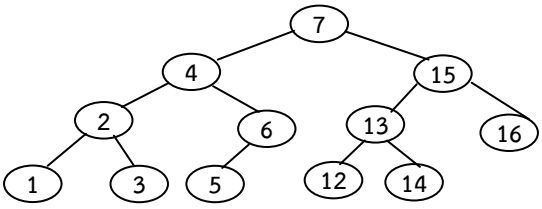
การแทรกโหนดใหม่ จะใช้หลักการเดียวกับการแทรกโหนดในไบนารีทรีเพื่อการค้นหา นั่นคือเปรียบเทียบค่าคีย์จากโหนดราก หากค่าคีย์ของโหนดใหม่น้อยกว่า จะไปยังโหนดลูกทางซ้าย แต่หากมากกว่าก็จะไปยังโหนดลูกทางขวา และเปรียบเทียบค่าคีย์ไปเรื่อยๆ จนกว่าจะพบตำแหน่งในการแทรกโหนดใหม่ ดังตัวอย่างในรูปที่ 10.11 เมื่อแทรกโหนดใหม่ (โหนด 11) ในโครงสร้างเอวีแอลทรีรูปที่ 10.11 (ก) จะได้ผลลัพธ์ของการดำเนินการแทรกโหนดใหม่ ดังรูปที่ 10.11 (ข)



(ก) เอวีแอลทรี



(ข) เอวีแอลทรีหลังการแทรกโหนด 11



(ค) เอวีแอลทรีหลังการปรับโครงสร้างทรี

รูปที่ 10.11 ตัวอย่างขั้นตอนการแทรกโหนดใหม่ในเอวีแอลทรี

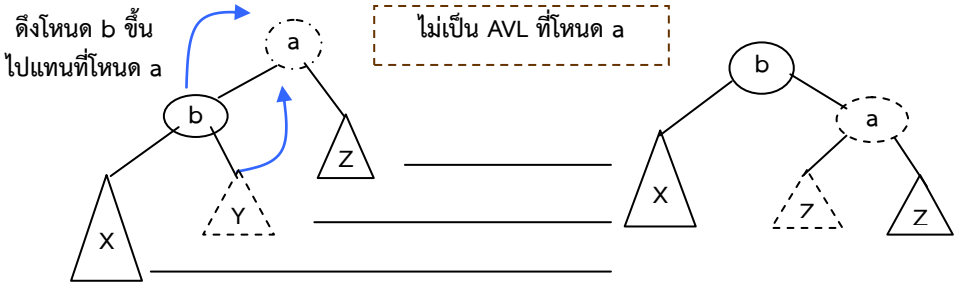
อย่างไรก็ตาม เนื่องจากเอวีแอลทรีเป็นโครงสร้างทรีที่มีลักษณะใกล้เคียงกับทรีแบบสมดุลมากที่สุด ทุกครั้งที่มีการดำเนินการเพื่อแทรกโหนดใหม่ในทรี จะมีการตรวจสอบว่าทรีขาดความสมดุลหรือไม่ ซึ่งพบว่าโหนด 15 เป็นตำแหน่งที่ทำให้ทรีขาดความเป็นเอวีแอลทรี เนื่องจากทรีย่อยทางซ้าย (โหนด 13) มีความสูงเท่ากับ 2 ในขณะที่ทรีย่อยทางขวา (โหนด 16) มีความสูงเท่ากับ 0 ดังรูปที่ 10.11 (ข) ซึ่งต้องแก้ไขด้วยการหมุนโหนด 13 มาแทนที่โหนด 15 ดังผลลัพธ์ในรูปที่ 10.11 (ค)

10.3.2 การหมุนทรี

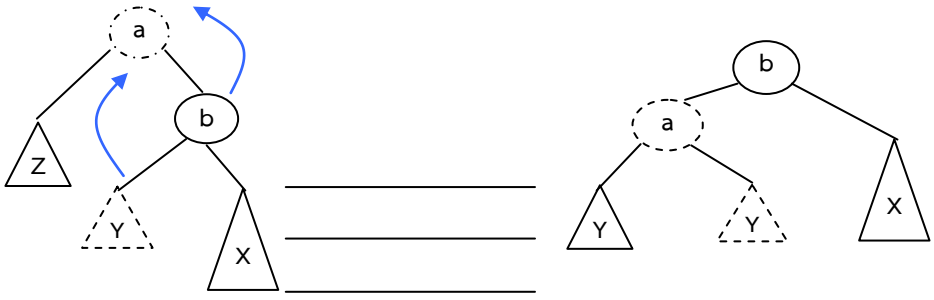
เมื่อไบนารีทรีขาดความเป็นเอวีแอลทรี จะแก้ไขด้วยการหมุนโหนดในทรีเพื่อปรับให้ทรีมีลักษณะเป็นเอวีแอลทรีเหมือนเดิม ซึ่งมี 2 รูปแบบ คือ การหมุนครั้งเดียว (Single Rotation) และการหมุนสองครั้ง (Double Rotation) มีรายละเอียดดังนี้

(1) การหมุนครั้งเดียว

การหมุนครั้งเดียวที่สามารถปรับโครงสร้างทรีให้มีความสมดุลตรงกับคุณสมบัติของเอวีแอลทรี ดังรูปที่ 10.12 พบว่าโหนด a คือ โหนดที่ทำให้ไบนารีทรีขาดคุณสมบัติความเป็นเอวีแอลทรี



(ก) การหมุนจากโหนดทางซ้ายไปขวา



(ข) การหมุนจากโหนดทางขวาไปซ้าย

รูปที่ 10.12 การหมุนครั้งเดียวเพื่อปรับความสมดุลของเอวีแอลทรี

วิธีการหมุนจะมี 2 ลักษณะดังนี้

1) การหมุนโหนดทางซ้ายไปขวา (Rotation With Left Node) จะใช้ในกรณีที่ทรีย่อยทางซ้ายของโหนด a มีความสูงมากกว่าทรีย่อยทางขวา จึงต้องลดระดับความสูงของทรีย่อยทางซ้ายของโหนด a ด้วยการหมุนโหนดลูกทางซ้าย (โหนด b) ให้ขึ้นไปอยู่ในตำแหน่งแทนที่โหนด a เดิม นั่นคือโหนดพ่อของ a จะมีโหนด b เป็นโหนดลูกแทน ดังรูปที่ 10.12 (ก) โดยกำหนดให้
   b คือ โหนดลูกทางซ้ายของโหนด a
   Y คือ ทรีย่อยทางขวาของโหนด b
   Z คือ ทรีย่อยทางขวาของโหนด a

วิธีการหมุนแบบนี้จะทำให้ต้องมีการเปลี่ยนการเชื่อมโยงของโหนด ดังนี้

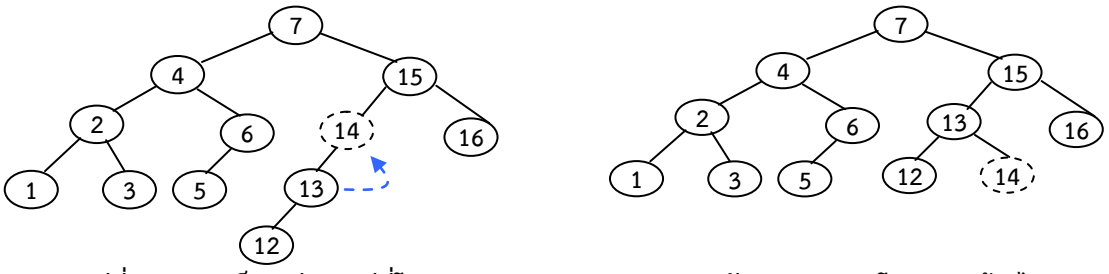
- ย้ายทรีย่อยทางขวาของโหนด b (หมายถึง Y) ไปเป็นทรีย่อยทางซ้ายของโหนด a เนื่องจากโหนดรากของทรีย่อย Y จะมีค่าข้อมูลมากกว่าโหนด b (เพราะอยู่ทางขวาของ b) แต่จะมีค่าน้อยกว่าโหนด a
- และย้ายโหนด a ไปเป็นโหนดลูกทางขวาของโหนด b

2) การหมุนโหนดทางขวาไปซ้าย (Rotation With Right Node) จะใช้ในกรณีที่ทรีย่อยทางขวาของโหนด a มีความสูงมากกว่าทรีย่อยทางซ้าย จึงต้องลดระดับความสูงของทรีย่อยทางขวาของโหนด a ด้วยการหมุนโหนดลูกทางขวา (โหนด b) ไปอยู่ในตำแหน่งแทนแสดงดังรูปที่ 10.12 (ข) โดยกำหนดให้

b คือ โหนดลูกทางขวาของโหนด a
Y คือ ทรีย่อยทางซ้ายของโหนด b
Z คือ ทรีย่อยทางซ้ายของโหนด a

วิธีการหมุนแบบนี้จะทำให้ต้องมีการเปลี่ยนการเชื่อมโยงของโหนด ดังนี้

- ย้ายทรีย่อยทางซ้ายของโหนด b (หมายถึงทรีย่อยY) ไปเป็นทรีย่อยทางขวาของโหนด a เนื่องจากโหนดรากของทรีย่อย Y จะมีค่าข้อมูลน้อยกว่าโหนด b (เพราะอยู่ทางซ้ายของ b) แต่จะมีค่ามากกว่าโหนด a
- และย้ายโหนด a ไปเป็นโหนดลูกทางซ้ายของโหนด b แทน



(ก) ทรีที่ขาดความเป็นเอวีแอลทรีที่โหนด 14 **(ข) หลังการหมุนจากโหนดทางซ้ายไปขวา**

รูปที่ 10.13 ตัวอย่างการหมุนจากโหนดทางซ้ายไปขวา

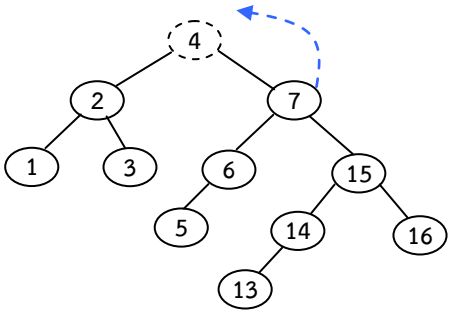
จากรูปที่ 10.13 แสดงตัวอย่างโครงสร้างไบนารีทรีที่พบว่าตำแหน่งโหนด 14 มีความสูงของทรีย่อยทางซ้ายเท่ากับ 1 ในขณะที่ทรีย่อยทางขวาเป็นทรีว่าง (มีความสูงเท่ากับ -1) จึงมีผลทำให้โหนด 14 เป็นโหนดที่ทำให้ไบนารีทรีขาดคุณสมบัติของความเป็นเอวีแอลทรี ดังรูปที่ 10. 13 (ก) และสามารถแก้ไขได้ด้วยวิธีการหมุนจากโหนดทางซ้ายไปขวาเนื่องจากทรีย่อยทางซ้ายมีความสูงมากกว่าด้วยการหมุนโหนด 13 ขึ้นไปแทนที่โหนด 14 ตามลำดับขั้นตอนดังนี้

1) ย้ายทรีย่อยทางขวาของโหนด 13 ไปเป็นทรีย่อยทางซ้ายของโหนด 14 ในที่นี้ คือ ทรีว่าง ดังนั้นโหนด 14 จึงไม่มีลูกทางซ้าย
2) ย้ายโหนด 14 ไปเป็นโหนดลูกทางขวาของโหนด 13

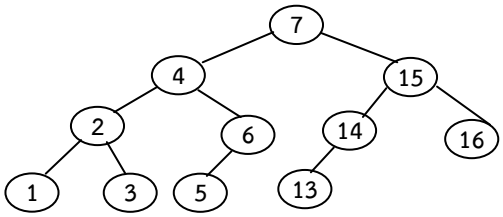
หลังจากการหมุนโหนด 13 ขึ้นไปแทนที่ โหนด 14 ทำให้แก้ปัญหาระดับความสูงของทรีย่อยของโหนด 14 ได้ และทำให้ทรีมีคุณสมบัติความเป็นเอวีแอลทรี ดังรูปที่ 10.13 (ข)

รูปที่ 10. 14 แสดงตัวอย่างโครงสร้างไบนารีทรีที่มีตำแหน่งโหนด 4 ทำให้ไบนารีทรีขาดคุณสมบัติของความเป็นเอวีแอลทรี และสามารถแก้ไขได้ด้วยวิธีการหมุนจากโหนดทางขวาไปซ้าย เนื่องจากโหนด 4 มีทรีย่อยทางขวามีความสูงมากกว่าทรีย่อยทางซ้าย เพื่อลดระดับความสูงของทรีย่อยทางขวาจึงต้องหมุนโหนด 7 ขึ้นไปแทนที่โหนด 4 แล้วย้ายโหนดอื่นๆ ดังนี้

1) ย้ายทรีย่อยทางซ้ายของโหนด 7 ไปเป็นทรีย่อยทางขวาของโหนด 4
2) ย้ายโหนด 4 ไปเป็นโหนดลูกทางซ้ายของโหนด 7



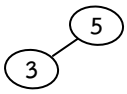
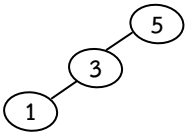
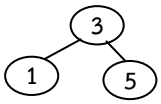
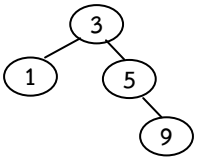
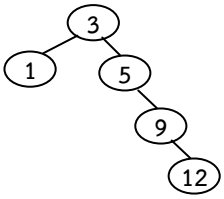
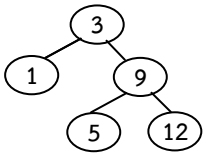
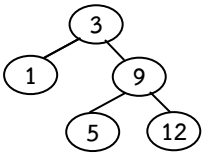
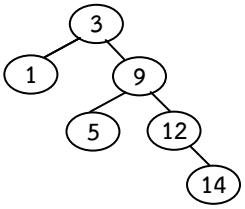
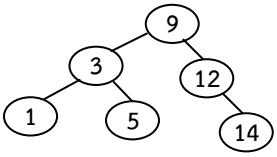
(ก) ทรีที่ขาดความเป็นเอวีแอลทรีที่โหนด 4



(ข) หลังการหมุนจากโหนดทางขวาไปซ้าย

รูปที่ 10.14 ตัวอย่างการหมุนจากโหนดทางขวาไปซ้าย

ตารางที่ 10.1 ตัวอย่างการปรับความสมดุลของเอวีแอลทรีหลังแทรกโหนดใหม่ด้วยการหมุนครั้งเดียว

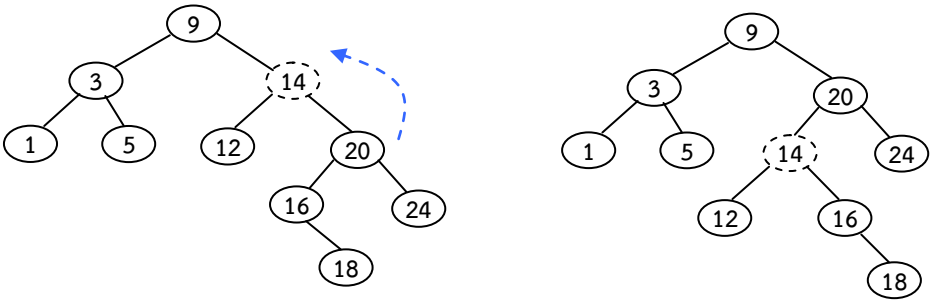
เอวีแอลทรี	ทรีที่ขาดคุณสมบัติเอวีแอลทรีหลังเพิ่มโหนดใหม่	เอวีแอลทรีหลังการหมุน
	 หลังเพิ่มโหนด 1 ขาดคุณสมบัติเอวีแอลทรีที่โหนด 5	 หลังหมุนโหนด 3 ขึ้น
	 หลังเพิ่มโหนด 12 ขาดคุณสมบัติเอวีแอลทรีที่โหนด 5	 หลังหมุนโหนด 9 ขึ้น
	 หลังเพิ่มโหนด 14 ขาดคุณสมบัติเอวีแอลทรีที่โหนด 3	 หลังหมุนโหนด 9 ขึ้น

จากตารางที่ 10.1 แสดงตัวอย่างการหมุนครั้งเดียวเพื่อแก้ไขปัญหาการขาดคุณสมบัติเอวีแอลทรีหลังการแทรกโหนดใหม่ในทรีมีรายละเอียดดังนี้

1) เมื่อแทรกโหนด 1 ในทรี แล้วทำให้ทรีขาดคุณสมบัติความเป็นเอวีแอลทรีที่โหนด 5 ซึ่งแก้ไขด้วยการหมุนจากโหนดซ้ายไปขวา โดยการหมุนโหนด 3 ขึ้นไปแทนที่โหนด 5 ดังแสดงในแถวที่ 1 ของตารางที่ 10.1
2) เมื่อแทรกโหนด 12 ในทรี แล้วทำให้ทรีขาดคุณสมบัติความเป็นเอวีแอลทรี ซึ่งแก้ไขด้วยการหมุนจากโหนดขวาไปซ้าย โดยการหมุนโหนด 9 ขึ้นไปแทนที่โหนด 5 ดังแสดงในแถวที่ 2 ของตารางที่ 10.1
3) เมื่อแทรกโหนด 14 ในทรี แล้วทำให้ทรีขาดคุณสมบัติความเป็นเอวีแอลทรีที่โหนด 3 ซึ่งแก้ไขด้วยการหมุนจากโหนดขวาไปซ้าย โดยการหมุนโหนด 9 ขึ้นไปแทนที่โหนด 3 ดังแสดงในแถวที่ 3 ของตารางที่ 10.1

อย่างไรก็ตาม ในบางกรณี วิธีการหมุนทรีเพียงครั้งเดียวอาจไม่สามารถแก้ปัญหาที่ทรีขาดคุณสมบัติความเป็นเอวีแอลทรีได้ ดังรูปที่ 10. 15 โครงสร้างทรีขาดคุณสมบัติความเป็นเอวีแอลทรีที่ตำแหน่งโหนด 14 ซึ่งทรีย่อยทางขวามีความสูงมากกว่าทรีย่อยทางซ้าย ดังรูปที่ 10. 15 (ก) และได้ทำการหมุนจากโหนดทางขวาไปทางซ้าย ดังแสดงในรูปที่ 10. 15 (ข) ด้วยการหมุนโหนด 20 ไปแทนที่โหนด 14 แล้วย้ายโหนดอื่นๆ ดังนี้

1) ย้ายทรีย่อยทางซ้ายของโหนด 20 ไปเป็นทรีย่อยทางขวาโหนด 14
2) ย้ายโหนด 14 ไปเป็นโหนดลูกทางซ้ายของโหนด 20



(ก) ทรีที่ขาดความเป็นเอวีแอลทรีที่โหนด 14 **(ข) หลังการหมุนจากโหนดทางขวาไปซ้าย**

รูปที่ 10.15 ตัวอย่างการหมุนครั้งเดียวแล้วไม่สามารถปรับความสมดุลของเอวีแอลทรีได้

(2) การหมุนสองครั้ง

จากรูปที่ 10.15 (ข) พบว่าหลังการหมุนทรีเพียงครั้งเดียว ไม่สามารถลดระดับความสูงของทรีได้ เพราะโครงสร้างทรีที่ได้หลังการหมุนขาดคุณสมบัติของความเป็นเอวีแอลทรีที่โหนด 20 แทนซึ่งสามารถแก้ไขปัญหานี้ได้ด้วยวิธีการหมุน 2 ครั้ง เมื่อพบเหตุการณ์ที่การหมุนครั้งเดียวไม่สามารถแก้ไขปัญหาการขาดคุณสมบัติของเอวีแอลทรีได้ วิธีการหมุนสองครั้งอาจหมุนจากซ้ายไปขวาแล้วค่อยหมุนจากขวาไปซ้ายอีกครั้งหรือสลับกันลักษณะการหมุนสองครั้งจะมี 2 แบบมีรายละเอียดดังนี้

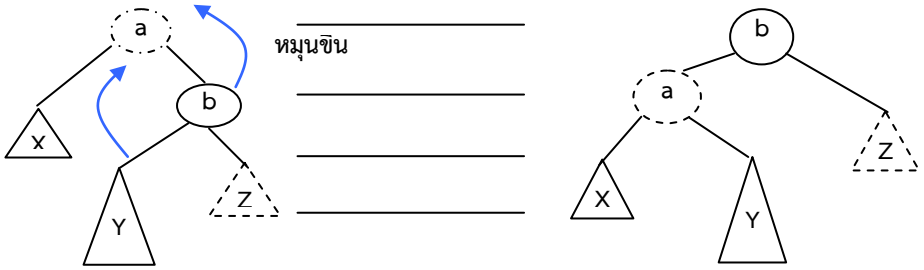
1) การหมุนสองครั้งด้วยโหนดลูกทางขวา

กำหนดให้โหนด a คือโหนดที่ทำให้ไบนารีทรีขาดคุณสมบัติความเป็นเอวีแอลทรี ดังรูปที่ 10.16 พบว่าทรีย่อยทางขวาของโหนด a มีความสูงมากกว่าทรีย่อยทางซ้าย หากหมุนโหนดลูกทางขวาของโหนด a คือ โหนด bขึ้นไปแทนที่โหนด aดังรูปที่ 10.16(ก) จะพบว่าจะทำให้โหนด b มีปัญหาเรื่องระดับความสูงของทรีย่อยทางซ้ายต่างจากทรีย่อยทางขวามากกว่า 1 มีผลให้ทรีขาดคุณสมบัติความเป็นเอวีแอลทรีที่โหนด b แทน แสดงว่าการหมุนครั้งเดียวไม่

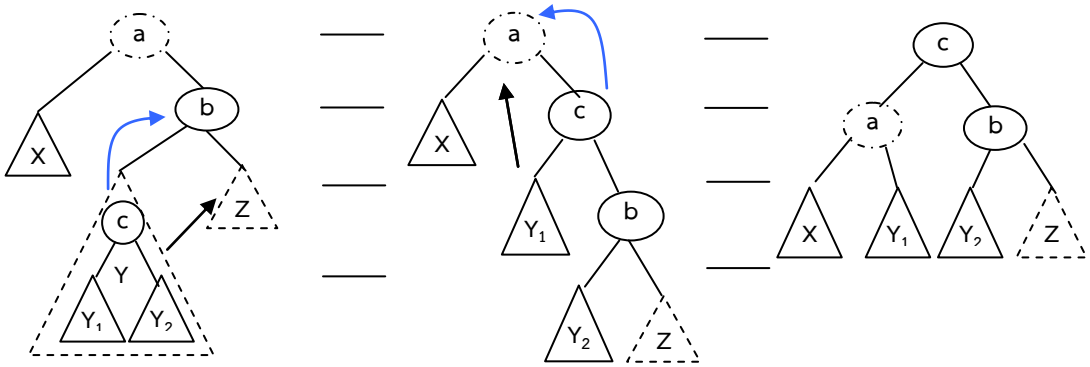
สามารถแก้ปัญหาการขาดคุณสมบัติความเป็นเอวีแอลทรีได้ ลักษณะนี้สามารถแก้ปัญหาของเหตุการณ์นี้ได้โดยการแยก Y ออกเป็น 2 ทรีย่อย Y1 และ Y2 โดยสมมติให้โหนด c เป็นโหนดรากของทรีย่อย Y ดังนั้น Y1 จะเป็นทรีย่อยทางซ้ายของโหนด c และ Y2 เป็นทรีย่อยทางขวาของโหนด c ดังรูปที่ 10.16(ข) จากนั้นทำการหมุนโหนด c จำนวน 2 ครั้งเพื่อลดระดับความสูงของทรีย่อย Y ดังนี้

- หมุนครั้งที่ 1 ด้วยการหมุนโหนด c ไปแทนที่โหนด b โดยใช้หลักการหมุนครั้งเดียวจากโหนดซ้ายไปขวา ซึ่งจะมีการย้ายการเชื่อมโยงของทรีดังนี้
    1) ย้ายทรีย่อยทางขวาของโหนด c ไปเป็นทรีย่อยทางซ้ายของโหนด b
    2) ย้ายโหนด b ไปเป็นโหนดลูกทางขวาของโหนด c
- จากนั้นหมุนครั้งที่ 2 ด้วยการหมุนโหนด c ไปแทนที่โหนด a โดยใช้หลักการหมุนครั้งเดียวจากโหนดขวาไปซ้าย ซึ่งจะมีการย้ายการเชื่อมโยงของทรีดังนี้
    1) ย้ายทรีย่อยทางซ้ายของโหนด c ไปเป็นทรีย่อยทางขวาของโหนด a
    2) ย้ายโหนด a ไปเป็นโหนดลูกทางซ้ายของโหนด c

หลังจากการหมุนโหนด c (โหนดรากของทรีย่อย Y) จำนวน 2 ครั้ง จะทำให้ระดับความสูงของทรีลดลง 1 ระดับ ซึ่งทำให้แก้ปัญหาการขาดคุณสมบัติความเป็นเอวีแอลทรีได้ ดังรูปที่ 10.16(ข)



(ก) ปัญหาการหมุนครั้งเดียวจากโหนดทางขวาไปซ้าย ระดับความสูงของ Y ไม่เปลี่ยน



(ข) การหมุนสองครั้งจากโหนดทางซ้ายไปขวา ตามด้วยการหมุนโหนดทางขวาไปซ้าย

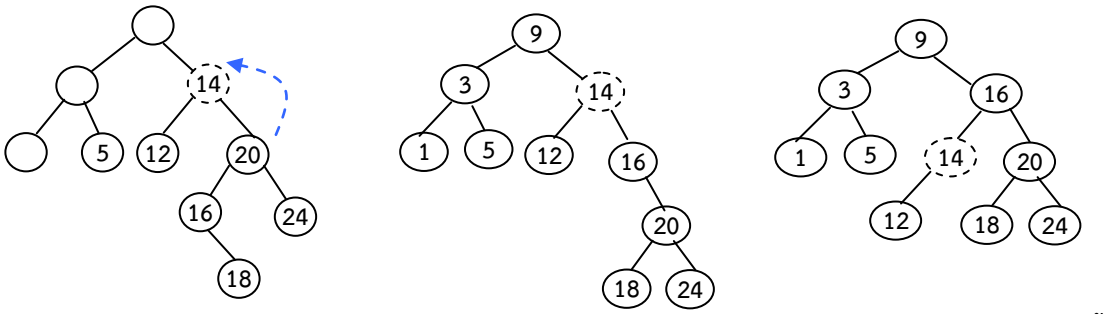
รูปที่ 10.16 การหมุนสองครั้งด้วยโหนดลูกทางขวา

รูปที่ 10.17 แสดงตัวอย่างการหมุนสองครั้งด้วยโหนดลูกทางขวา เนื่องจากรูปที่ 10. 17 (ก) ตำแหน่งโหนด 14 ทำให้โครงสร้างไบนารีทรีขาดคุณสมบัติของความเป็นเอวีแอลทรี และต้องแก้ไขด้วยการลดระดับความสูงของทรีย่อยทางขวา (ที่ตำแหน่งโหนด 20) ของโหนด 14 ซึ่งพบว่าทรีย่อย

ทางซ้าย (มีโหนด 16 เป็นโหนดราก) ของโหนด 20 มีความสูงมากกว่าทรีย่อยทางขวา ดังนั้นจะใช้วิธีการหมุนโหนด 16 จำนวน 2 ครั้ง เพื่อลดระดับความสูงของทรี มีรายละเอียดดังนี้

- ครั้งที่ 1 ให้หมุนโหนดลูกทางซ้าย (โหนด 16) ของโหนด 20 ด้วยการหมุนโหนด 16 ไปแทนที่โหนด 20 ดังรูปที่ 10.17 (ข)โดยต้องย้ายโหนดอื่นๆ ดังนี้
  1) ย้ายทรีย่อยทางขวาของโหนด 16 ไปเป็นทรีย่อยทางซ้ายของโหนด 20
  2) ย้ายโหนด 20 ไปเป็นโหนดลูกทางขวาของโหนด 16
- ครั้งที่ 2 ให้หมุนโหนดลูกทางขวา (โหนด 1 6) ของโหนด 14 ด้วยการหมุนโหนด 16 ไปแทนที่โหนด 14 ดังรูปที่ 10.17 (ค)
  1) ย้ายทรีย่อยทางซ้ายของโหนด 16 ไปเป็นทรีย่อยทางขวาของโหนด 14
  2) ย้ายโหนด 14 ไปเป็นโหนดลูกทางซ้ายของโหนด 16

หลังจากการหมุนโหนด 16 จำนวน 2 ครั้ง จะพบว่าความสูงของทรีลดลง 1 ระดับทำให้แก้ปัญหาการขาดคุณสมบัติความเป็นเอวีแอลทรีของโหนด 14 ได้



(ก) หมุนโหนด 16 ไปแทนที่โหนด 20 **(ข) หมุนโหนด 16 ไปแทนที่โหนด 14** **(ค) หลังหมุนโหนด 16 จำนวน 2 ครั้ง**

รูปที่ 10.17 ตัวอย่างการหมุนสองครั้งด้วยโหนดลูกทางขวา

2) การหมุนสองครั้งด้วยโหนดลูกทางซ้าย

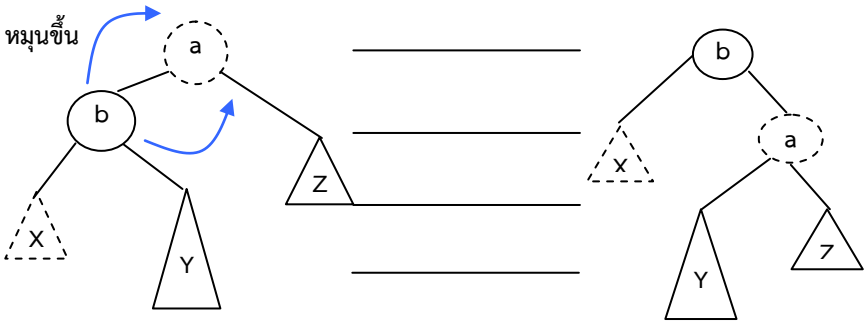
จากรูปที่ 10.18 แสดงตัวอย่างโครงสร้างทรีกรณีทรีย่อยทางซ้ายของโหนด a มีความสูงมากกว่าทรีย่อยทางขวาและพบว่าเมื่อหมุนโหนดลูกทางขวา (โหนด b) ของโหนด a ขึ้นไปแทนที่โหนด a ดังรูปที่ 10.18 (ก) ก็ไม่สามารถแก้ปัญหาการขาดคุณสมบัติความเป็นเอวีแอลทรีได้เนื่องจากความสูงของทรีย่อยทางขวาของโหนด b (ทรีย่อย Y) ยังคงมีระดับความสูงเท่าเดิมการหมุนโหนด b ไปแทนโหนด a จึงทำให้ทรีขาดคุณสมบัติความเป็นเอวีแอลทรีที่โหนด b แทน

ในกรณีนี้สามารถแก้ปัญหาด้วยการหมุนสองครั้งจากโหนดลูกทางซ้าย โดยจะแยก Y ออกเป็น 2 ทรีย่อย Y1 และ Y2 โดยสมมุติให้โหนด c เป็นโหนดรากของทรีย่อย Y ดังนั้น Y1 จะเป็นทรีย่อยทางซ้ายของโหนด c และ Y2 เป็นทรีย่อยทางขวาของโหนด c ดังรูปที่ 10.18 (ข) จากนั้นทำการหมุนโหนด c จำนวน 2 ครั้งเพื่อลดระดับความสูงของทรีย่อย Y ดังนี้

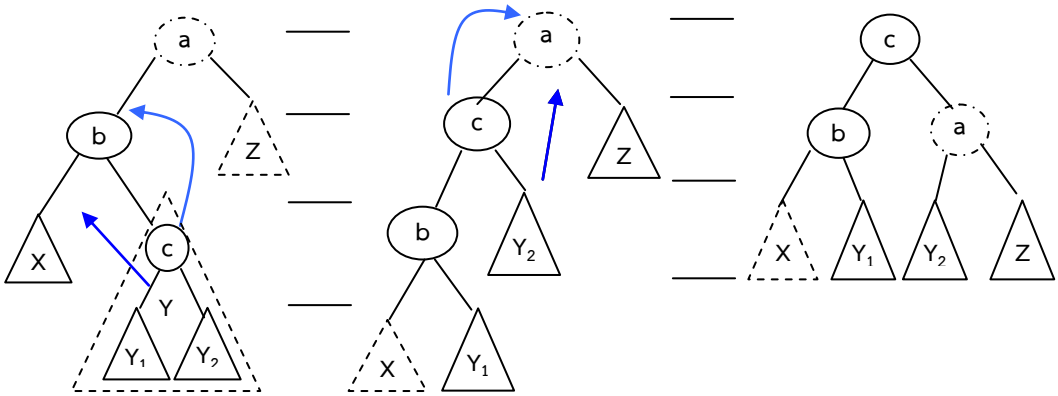
- หมุนครั้งที่ 1 ด้วยการหมุนโหนด c ไปแทนที่โหนด b โดยใช้หลักการหมุนครั้งเดียวจากโหนดขวาไปซ้าย ซึ่งจะมีการย้ายการเชื่อมโยงของทรีดังนี้
  1) ย้ายทรีย่อยทางซ้ายของโหนด c ไปเป็นทรีย่อยทางขวาของโหนด b
  2) ย้ายโหนด b ไปเป็นโหนดลูกทางซ้ายของโหนด c

- จากนั้นหมุนครั้งที่ 2 ด้วยการหมุนโหนด c ไปแทนที่โหนด a โดยใช้หลักการหมุนครั้งเดียวจากโหนดซ้ายไปขวา ซึ่งจะมีการย้ายการเชื่อมโยงของทรี ดังนี้
  1) ย้ายทรีย่อยทางขวาของโหนด c ไปเป็นทรีย่อยทางซ้ายของโหนด a
  2) ย้ายโหนด a ไปเป็นโหนดลูกทางขวาของโหนด c

หลังจากการหมุนโหนด c (โหนดรากของทรีย่อย Y) จำนวน 2 ครั้ง ระดับความสูงของทรีลดลง 1 ระดับ และสามารถแก้ปัญหาการขาดคุณสมบัติความเป็นเอวีแอลทรีได้ ดังรูปที่ 10.18 (ข)



(ก) ปัญหาการหมุนครั้งเดียวจากโหนดทางซ้ายไปขวา ระดับความสูงของ Y ไม่เปลี่ยน

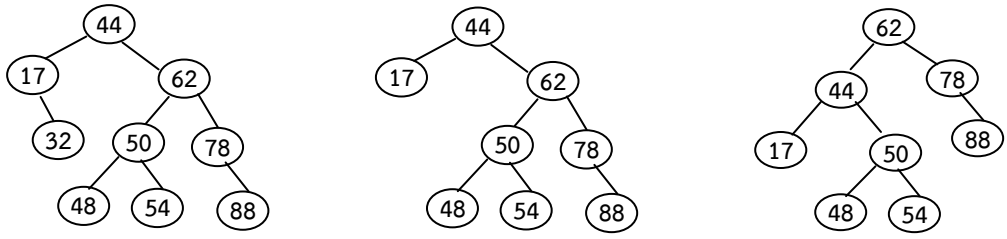


(ข) การหมุนสองครั้งจากโหนดทางขวาไปซ้าย ตามด้วยการหมุนโหนดทางซ้ายไปขวา

รูปที่ 10.18 การหมุนสองครั้งด้วยโหนดลูกทางซ้าย

10.3.3 การลบโหนดในเอวีแอลทรี

การลบโหนดในเอวีแอลทรีจะใช้วิธีเดียวกับการลบโหนดในไบนารีทรีเพื่อการค้นหา และหลังจากการลบโหนดก็อาจทำให้โครงสร้างทรีขาดคุณสมบัติความเป็นเอวีแอลทรีได้เช่นกัน ดังรูปที่ 10.19 (ก) หลังจากลบโหนด 32 ออกจากโครงสร้างทรีจะได้รูปทรีดังรูปที่ 10. 19(ข) ซึ่งพบว่าทำให้โครงสร้างทรีขาดคุณสมบัติความเป็นเอวีแอลทรีที่โหนด 44 และสามารถแก้ไขได้ด้วยหลักการหมุนครั้งเดียวจากโหนดขวาไปซ้าย โดยการหมุนโหนดลูกทางขวา (โหนด 62) มาแทนที่โหนด 44 เนื่องจากทรีย่อยทางขวามีความสูงมากกว่าทรีย่อยทางซ้ายของโหนด 44 ดังรูปที่ 10.19 (ค)



(ก) ทรีก่อนการลบโหนด 32 **(ข) ทรีหลังการลบโหนด 32** **(ค) หลังหมุนโหนด 62 ไปแทนที่โหนด 44**

รูปที่ 10.19 การลบโหนดและการหมุนครั้งเดียวจากโหนดขวาไปซ้าย

10.4 การแวะผ่านทรี

การแวะผ่านทรี (Tree Traversal) คือ การเข้าถึง (Access) หรือ การเยี่ยม (Visit) โหนดในทรี ซึ่งการแวะผ่านทรีอย่างมีประสิทธิภาพ จะ ทำให้สามารเข้าถึงโหนดแบบไม่ซ้ำกันได้อย่างรวดเร็วและเชื่อถือได้ว่าสามารถประมวลผลข้อมูลได้ครบทุกโหนดในทรี

10.4.1 การเข้าถึงโหนดในทรี

วิธีการเข้าถึงโหนดต่างๆ ในทรีแบ่งเป็น 2 รูปแบบ ดังนี้

1) การแวะผ่านทรีในไบนารีทรีแบบแนวกว้าง (Breadth-first)มีลักษณะการเข้าถึงแต่ละโหนดในทรีทีละระดับ โดยเข้าถึงโหนดทั้งหมดในระดับแรกก่อนจึงเข้าถึงทุกโหนดในระดับถัดไป จนครบทุกระดับในโครงสร้างทรี การเข้าถึงข้อมูลแบบนี้สามารถนำโครงสร้างคิวมาช่วยในการเก็บข้อมูลมีขั้นตอนการเข้าถึงข้อมูลดังนี้

```
breadth-first(q) {
    q.enqueue(p)
    while (!q.isEmpty())
        q.dequeue(p)
        Print item in node p
        for each child w of p
            q.enqueue(w)
}
```

2) การแวะผ่านทรีในไบนารีทรีแบบแนวลึก (Depth-first)แบ่งเป็น 2 ลักษณะคือ

2.1) การเข้าถึงแบบพรีออร์เดอร์ (Preorder Traversal)

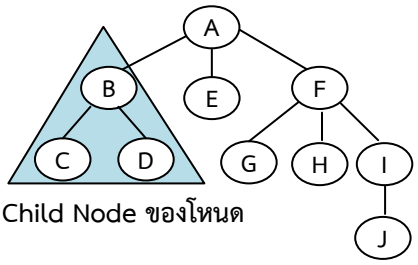
ข้อมูลของแต่ละโหนดในทรี จะถูกเข้าถึงหรือถูกประมวลผลก่อนโหนดลูกหลานของตนเอง มีอัลกอริทึมในการดำเนินการ ดังนี้

```
preOrder(p){
    print item in node p
    for each child w of p
        preOrder (w)
}
```

2.2) การเข้าถึงแบบโพสต์ออร์เดอร์ (Postorder Traversal)

ข้อมูลของแต่ละโหนดในทรีจะถูกเข้าถึงหรือถูกประมวลผลหลังจากการเข้าถึงข้อมูลของโหนดลูกหลานของตนอง มีอัลกอริทึมในการดำเนินการ ดังนี้

```
postOrder(p){
    for each child w of p
        postOrder (w)
    print item in node p
}
```



รูปที่ 10.20 ตัวอย่างโครงสร้างข้อมูลทรี

จากตัวอย่างทรีในรูปที่ 10.20 ลำดับการเข้าถึงแต่ละโหนดในทรีแบบแนวกว้างคือ A B E F C D G H I J ส่วนลำดับการเข้าถึงโหนดแบบพรีออร์เดอร์ คือ A B C D E F G H I J และลำดับการเข้าถึงแต่ละโหนดในทรีแบบโพสต์ออร์เดอร์ คือ C D B E G H J I F A

10.4.2 การเข้าถึงโหนดในไบนารีทรี

การแวะผ่านทรีในไบนารีทรี (Traversing Binary Tree) สามารถทำได้ 4 วิธี แต่ละวิธีแตกต่างกันตรงที่จุดเริ่มต้นในการแวะผ่านทรีว่าจะเริ่มต้นที่โหนดราก หรือ ทรีย่อยทางซ้าย (T_L) หรือทรีย่อยทางขวา (T_R) ก่อน หรือจะเข้าถึงโหนดข้อมูลในแต่ละระดับของทรี แต่ละวิธีมีรายละเอียดดังนี้

1) การแวะผ่านทรีในไบนารีทรีแบบอินออร์เดอร์ (Inorder Traversal) มีลักษณะการเข้าถึงแบบ T_L R T_R คือ จะเข้าถึงทุกโหนดในทรีย่อยทางขวาของโหนดใดๆ ก่อน จึงจะเข้าถึงข้อมูลของโหนดนั้น แล้วตามด้วยทุกโหนดทรีย่อยทางซ้ายของโหนดนั้น ตามลำดับ มีขั้นตอนการทำงานดังนี้

```
inOrder(p) {
    if hasLeft (p)
        inOrder (leftChild (p))
     Print item in node p
    if hasRight (p)
        inOrder (rightChild (p))
}
```

2) การแวะผ่านทรีในไบนารีทรีแบบพรีออร์เดอร์ (Preorder Traversal)มีลักษณะการเข้าถึงแบบ R T_L T_R คือ จะเข้าถึงข้อมูลของโหนดใดๆ ก่อนจึงท่องไปยังทุกโหนดในทรีย่อยทางขวาและตามด้วยทุกโหนดทรีย่อยทางซ้ายของโหนดนั้น มีขั้นตอนการทำงานดังนี้

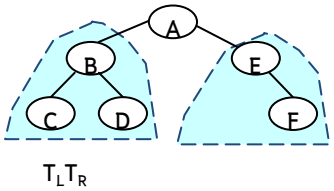
```
preOrder(p) {
    Print item in node p
    if hasLeft (p)
        preOrder (leftChild (p))
    if hasRight (p)
        preOrder (rightChild (p))
}
```

3) การแวะผ่านทรีในไบนารีทรีแบบโพสต์ออร์เดอร์ (Postorder Traversal)มีลักษณะการเข้าถึงแบบ T_L T_RR คือ จะเข้าถึงทุกโหนดในทรีย่อยทางขวาของโหนดใดๆ ก่อน ตามด้วยทุกโหนดทรีย่อยทางซ้ายของโหนดนั้น แล้วจึงจะเข้าถึงข้อมูลของโหนดนั้น มีขั้นตอนการทำงานดังนี้

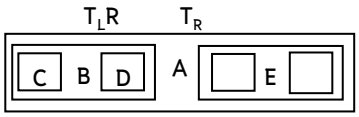
```
postOrder(p) {
    if hasLeft (p)
        postOrder (leftChild (p))
    if hasRight (p)
        postOrder (rightChild (p))
    Print item in node p
}
```

4) การแวะผ่านทรีในไบนารีทรีแบบแนวกว้าง (Breadth-first)มีลักษณะการเข้าถึงแต่ละโหนดในทรีทีละระดับเช่นเดียวกับทรีทั่วไป

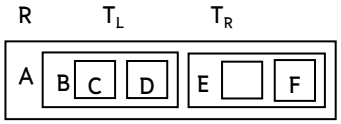
รูปที่ 10.21 แสดงตัวอย่างไบนารีทรีในรูปที่ 10.21 (ก) ซึ่งแบ่งเป็นทรีย่อยทางซ้ายและทรีย่อยทางขวา ผลการเข้าถึงข้อมูลในไบนารีทรีแบบอินออร์เดอร์ แบบพรีออร์เดอร์ แบบโพสต์ออร์เดอร์ และแบบแนวกว้างแสดงดังรูปที่ 10.21 (ข) – (จ) ตามลำดับ



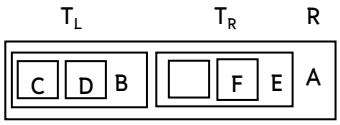
(ก) โครงสร้างข้อมูลไบนารีทรี



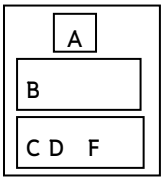
(ข) ผลการเข้าถึงแบบอินออร์เดอร์



(ค) ผลการเข้าถึงแบบพรีออร์เดอร์



(ง) ผลการเข้าถึงแบบโพสต์ออร์เดอร์

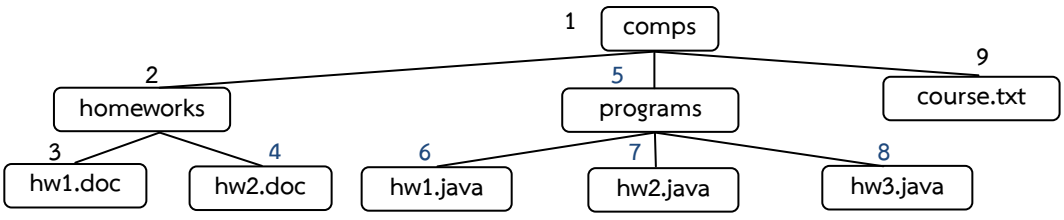


(จ) ผลการเข้าถึงแบบแนวกว้าง

รูปที่ 10.21 การเข้าถึงข้อมูลในไบนารีทรี

10.4.3 การประยุกต์ใช้งานการแวะผ่านทรี

การแวะผ่านทรีสามารถถูกนำไปประยุกต์ใช้เพื่อการค้นหาและแสดงผลข้อมูลในทรี เช่น การแสดงโครงสร้างการจัดเก็บข้อมูลของไฟล์และโฟลเดอร์ระบบการจัดการไฟล์ (File System) ในคอมพิวเตอร์ ดังรูปที่ 10.22



รูปที่ 10.22 โครงสร้างการจัดเก็บของระบบไฟล์ในคอมพิวเตอร์และลำดับการเข้าถึงแบบพรีออร์เดอร์

เมื่อเข้าถึงข้อมูลแบบพรีออร์เดอร์ และแบบโพสต์ออร์เดอร์จะสามารถแสดงโครงสร้างการจัดเก็บไฟล์และโฟล์เดอร์ได้ดังนี้

ผลลัพธ์การเข้าถึงแบบพรีออร์เดอร์

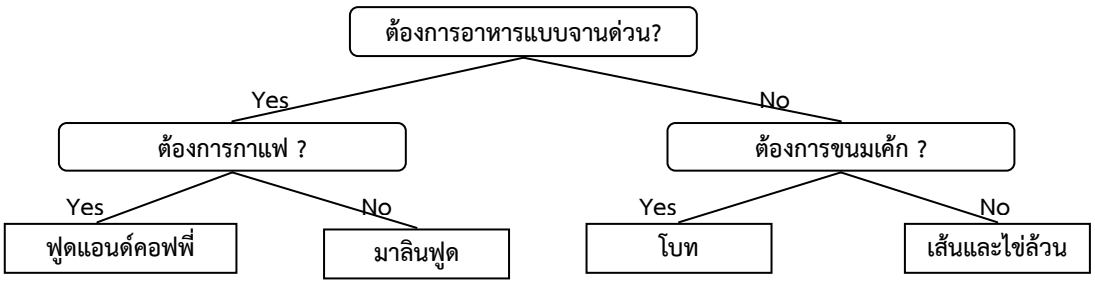
comps
homeworks
    hw1.doc
    hw2.doc
programs
    hw1.java
    hw2.java
    hw3.java
course.txt

ผลลัพธ์การเข้าถึงแบบพรีออร์เดอร์

hw1.doc
hw2.doc
homeworks
    hw1.java
    hw2.java
    hw3.java
programs
  course.txt
comps

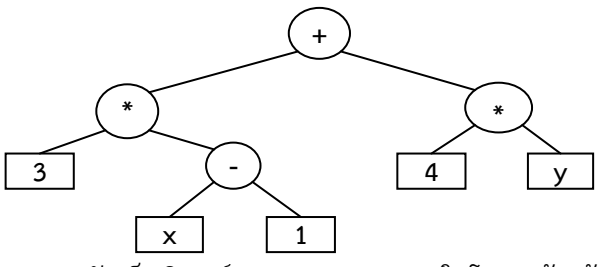
สำหรับตัวอย่างการนำโครงสร้างไบนารีทรีไปประยุกต์ใช้งาน เช่น การสร้างทรีสำหรับการตัดสินใจ (Decision Tree) โดยโหนดภายใน (Internal Nodes) ประกอบด้วยประโยคคำถามให้ตอบว่า Yes หรือ No โดยแต่ละคำถาม หากตอบ Yes ให้ไปยังคำถามต่อไปที่โหนดลูกทางซ้าย และหากตอบ No ให้ไปยังคำถามต่อไปที่โหนดลูกทางขวา เมื่อเข้าถึงโหนดใบหรือโหนดภายนอก จะเป็นการสิ้นสุดการตัดสินใจ ดังตัวอย่างในรูปที่ 10.23 เป็นตัวอย่างไบนารีทรีสำหรับการตัดสินใจเลือกสถานที่สำหรับรับประทานอาหาร

รูปที่ 10.23 หากตอบ Yes ที่คำถาม “ต้องการอาหารแบบจานด่วน” จะไปยังคำถาม “ต้องการดื่มกาแฟ?” ซึ่งหากตอบ Yes ที่คำถามนี้ ก็จะสิ้นสุดการตัดสินใจเลือกสถานที่รับประทานอาหารเป็นร้าน ฟูดแอนด์คอฟฟี่เป็นต้น



รูปที่ 10.23 ตัวอย่างไบนารีทรีสำหรับการตัดสินใจเลือกสถานที่สำหรับรับประทานอาหาร

รูปที่ 10.24 แสดงตัวอย่างการประยุกต์ใช้โครงสร้างข้อมูลไบนารีทรีในการจัดเก็บประโยคคำนวณทางคณิตศาสตร์ของนิพจน์ (3*(x-1) + (4*y)) โดยโหนดภายใน (Internal Nodes) ใช้สำหรับจัดเก็บการดำเนินการทางคณิตศาสตร์ของเครื่องหมาย + - * และ / ส่วนโหนดใบจะเป็นตัวถูกดำเนินการ (Operands) ซึ่งอาจเป็นชื่อตัวแปร หรือค่าคงที่สำหรับใช้ในการคำนวณ



รูปที่ 10.24 การจัดเก็บนิพจน์ (3*(x-1) + (4*y)) ในโครงสร้างข้อมูลไบนารีทรี

จากรูปที่ 10.24 การเข้าถึงโหนดในทรีแบบพรีออร์เดอร์ สามารถแสดงค่านิพจน์ในลักษณะแบบ prefix คือ +*3-x1*4y และหากเข้าถึงโหนดในทรีแบบโพสต์ออร์เดอร์ จะได้นิพจน์แบบ postfix คือ 3 x1-*4y*+ โดยนิพจน์ทั้งแบบ prefix และ postfix ที่ได้จะสามารถแปลงเป็นนิพจน์ (3*(x-1)+(4*y)) โดยใช้สแตกช่วยในการจัดเก็บข้อมูล อย่างไรก็ตามหากเข้าถึงโหนดในทรีแบบอินออร์เดอร์ ก็สามารถแสดงนิพจน์ในลักษณะแบบ infix ได้ คือ 3* x-1+4*y เพียงแต่นิพจน์ที่ได้ยังมีลำดับในการคำนวณที่ผิดจึงต้องใส่วงเล็บเพิ่มระหว่างการเข้าถึงโหนดเพิ่มเติมด้วย

นอกจากนี้ การจัดเก็บข้อมูลนิพจน์ทางการคำนวณในไบนารีทรียังสามารถถูกนำไปประยุกต์ใช้เพื่อการดำเนินการการคำนวณหาค่านิพจน์ได้ โดยมีขั้นตอนในการทำงานดังนี้

```
ItemType evalExpr (BinaryTree p) {
    var a, b : ItemType
    if isExternal (p)
        return p.getItem()
    else
        a← evalExpr (leftChild(p))
        b← evalExpr (rightChild(p))
        x ← p.getItem ( ) //operator stored at node p
        return Compute(a, x, b)
}
```

10.5 การนำทรีไปใช้

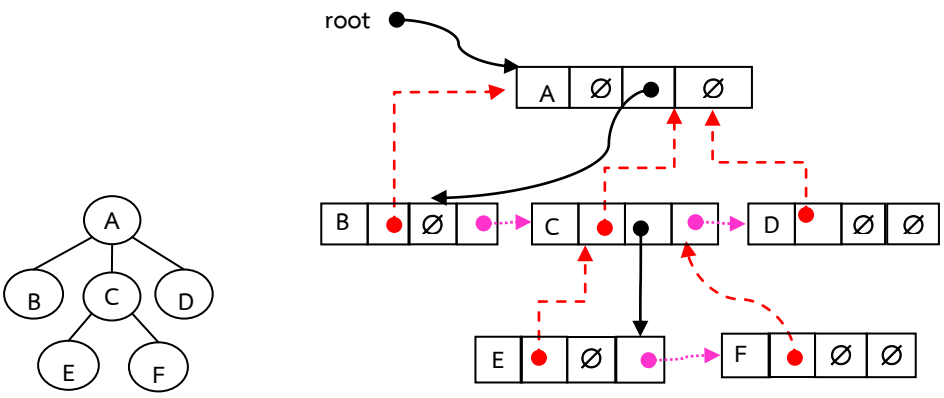
การนำทรีไปใช้ในคอมพิวเตอร์ด้วยการเขียนเป็นภาษาคอมพิวเตอร์ สามารถสร้างด้วยอาร์เรย์หรือลิงค์ลิสต์ก็ได้

10.5.1 การออกแบบการสร้างทรี

โครงสร้างข้อมูลทรีที่สร้างด้วยลิงค์ลิสต์ แต่ละสมาชิกจะถูกเชื่อมโยงกันด้วยโหนดข้อมูล ดังรูปที่ 10.25 จากข้อมูลในโครงสร้างทรีรูปที่ 10.25 (ก) ประกอบด้วยจำนวนสมาชิกทั้งหมด 5 ค่า คือ A B C D และ E สามารถออกแบบการจัดเก็บข้อมูลด้วยลิงค์ลิสต์ได้ดังรูปที่ 10. 25 (ข) แต่ละโหนดข้อมูลประกอบด้วย item ที่เก็บค่าสมาชิกของทรี ตัวชี้ parent ทำหน้าที่เชื่อมโยงไปยังโหนดพ่อ ตัวชี้firstChild ทำหน้าที่เชื่อมโยงไปยังโหนดลูกที่เป็นโหนดแรกของลิสต์ที่เก็บรายการเชื่อมโยงของโหนดลูก และตัวชี้ sibling ทำหน้าที่เชื่อมโยงไปยังโหนดถัดไปที่เป็นโหนดพี่น้องกัน

ตัวอย่างการออกแบบโครงสร้างโหนดของทรีด้วยคำสั่งภาษาซี มีดังนี้

```
typedef  struct node_content {
    itemType item;
    struct node_content *parent;
    struct node_content *firstChild;
    struct node_content *sibling;
} node_type;
root node_type  *root;
```



(ก) ตัวอย่างโครงสร้างทรี (ข) ตัวอย่างโครงสร้างทรีที่สร้างด้วยลิงค์ลิสต์

รูปที่ 10.25 ตัวอย่างการสร้างโครงสร้างทรีด้วยลิงค์ลิสต์

จากรูปที่ 10.25 (ข) สามารถอธิบายตัวอย่างรายละเอียดการจัดเก็บข้อมูลในโหนดได้ดังนี้

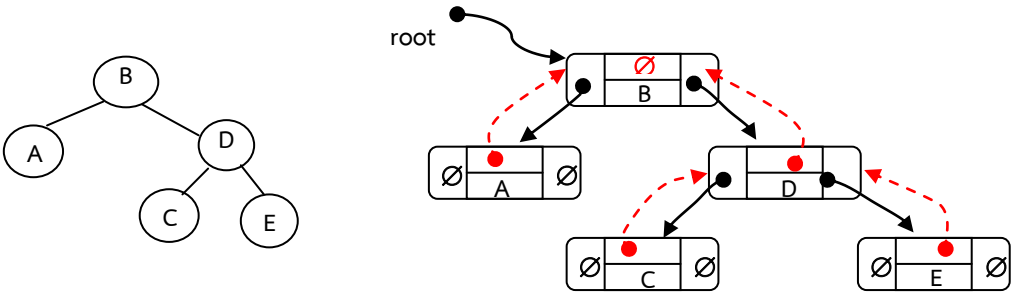
- โหนดแรกของทรีจะมีตัวชี้ชื่อ root ทำหน้าที่เชื่อมโยงไปยังโหนดราก (โหนด A) ของทรี
- ลักษณะของโหนด B มีดังนี้
  - ตัวชี้ parent และตัวชี้ sibling จึงเป็นค่า Null เนื่องจากโหนด A เป็นโหนดรากของทรี
  - ตัวชี้ firstChild ของโหนด A จะเชื่อมโยงไปยังโหนด B ซึ่งเป็นโหนดลูกโหนดแรกในลิสต์ B C และ D ซึ่งทั้ง 3 โหนดนี้จึงมีตัวชี้ parent ชี้ไปยังโหนด A เหมือนกัน
- ลักษณะของโหนด D มีดังนี้
  - ตัวชี้ parent เชื่อมโยงไปยังโหนด A เป็นโหนดลูกของ A
  - ตัวชี้ sibling จะเชื่อมโยงไปค่า Null เนื่องจากเป็นโหนดสุดท้ายในลิสต์โหนดพี่น้อง คือ B C และD
  - ตัวชี้ firstChild จะเป็นค่า Null เนื่องจากโหนด D ไม่มีโหนดลูก

10.5.2 การออกแบบการสร้างไบนารีทรี

การสร้างโครงสร้างข้อมูลไบนารีทรีด้วยลิงค์ลิสต์ แต่ละสมาชิกของทรีจะถูกเชื่อมโยงกันด้วยโหนดข้อมูล ดังรูปที่ 10. 26 แสดงตัวอย่างการออกแบบทรีด้วยลิงค์ลิสต์ โดยแต่ละโหนดข้อมูลประกอบด้วย item ที่เก็บค่าสมาชิกของทรี ตัวชี้ parent ทำหน้าที่เชื่อมโยงไปยังโหนดพ่อ ตัวชี้ left ที่ทำหน้าที่เชื่อมโยงไปยังโหนดลูกทางซ้าย และ right ที่ทำหน้าที่เชื่อมโยงไปยังโหนดลูกทางขวา เนื่องจากเป็นไบนารีทรีโหนดลูกจึงเหลือเพียงโหนดลูกทางซ้ายและโหนดลูกทางขวาเท่านั้น และหากโหนดใดไม่มีโหนดลูกก็จะกำหนดตัวชี้ของโหนดลูกเป็นค่า Null ในการสร้างทรีจะมีการกำหนดตัวชี้ชื่อ root เพื่อทำหน้าที่เชื่อมโยงไปยังโหนดราก (โหนด B) ของทรีซึ่งโหนดรากของทรีจะไม่มีโหนดพ่อ ดังนั้นโหนดรากจะมีค่าตัวชี้ parent เป็นค่า Null เสมอ ดังรูปที่ 10.26 (ข) แสดงตัวอย่างการสร้างไบนารีทรีด้วยลิงค์ลิสต์ที่ประกอบด้วยจำนวนสมาชิกทั้งหมด 5 ค่า คือ A B C D และ E ดังรูปที่ 10.26 (ก)

ตัวอย่างการออกแบบโครงสร้างโหนดของทรีด้วยคำสั่งภาษาซี มีดังนี้

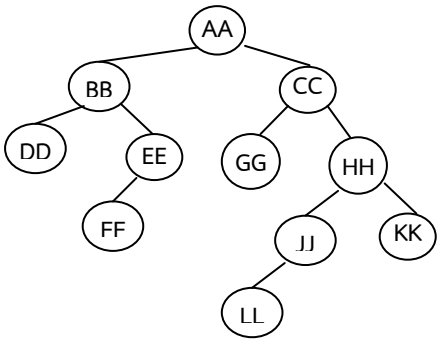
```
typedef  struct node_content {
        itemType item;
        struct node_content *parent;
        struct node_content *left, *right;
} node_type;
node_type  *root;
```



(ก) ตัวอย่างโครงสร้างไบนารีทรี (ข) การจัดเก็บข้อมูลไบนารีทรีที่สร้างด้วยลิงค์ลิสต์

รูปที่ 10.26 ตัวอย่างการสร้างโครงสร้างไบนารีทรีด้วยลิงค์ลิสต์

นอกจากนี้ยังสามารถออกแบบการสร้างทรีด้วยอาร์เรย์ ดังตัวอย่างในรูปที่ 10.27 แสดงตัวอย่างโครงสร้างไบนารีทรีที่ประกอบด้วย 11 โหนด ดังรูปที่ 10. 27 (ก) มีโหนด AA เป็นโหนดราก สามารถสร้างการจัดเก็บข้อมูลทรีด้วยอาร์เรย์ได้ดังรูปที่ 10.27 (ข)



(ก) ตัวอย่างโครงสร้างข้อมูลไบนารีทรีจำนวน 11 โหนด

	item	left	right
1	AA	2	3
2	BB	4	5
3	CC	7	8
4	DD	0	0
5	EE	6	0
6	FF	0	0
7	GG	0	0
8	HH	9	10
9	JJ	11	0
10	KK	0	0
11	LL	0	0
12		13	0
13		14	
14		15	
15		0	

(ข) การจัดเก็บข้อมูลไบนารีทรีด้วยอาร์เรย์

รูปที่ 10.27 ตัวอย่างการสร้างโครงสร้างไบนารีทรีด้วยอาร์เรย์

ตัวอย่างการออกแบบโครงสร้างโหนดในการจัดเก็บโหนดของทรีในอาร์เรย์มีดังนี้

```
struct binary_node {
    itemType item[10];
    int left, right;
} node_type;
node_type tree[16];
int root, avail;
```

โครงสร้างทรีที่ออกแบบ ด้วยอาร์เรย์ข้างต้น สามารถจัดเก็บข้อมูลในทรีได้จำนวนทั้งหมด 16 โหนด โดยโหนดรากจัดเก็บที่อาร์เรย์ tree[1] ดังนั้นตัวแปร root จะเก็บค่า 1 เนื่องจากโหนดราก (โหนด AA) มีโหนดลูกทางซ้ายคือโหนด BB และ โหนดลูกทางขวาคือ โหนด CC ซึ่งจัดเก็บในอาร์เรย์ tree[2] และ tree[3] ตามลำดับ ดังนั้นค่า left และ right ในอาร์เรย์ tree[1] จึงมีค่าเท่ากับ 2 และ 3 ตามลำดับ ซึ่งหมายถึงตำแหน่งในการเก็บค่าโหนดลูกทางซ้ายและโหนดลูกทางขวาของโหนด AA ดังแสดงตัวอย่างในรูปที่ 10.27 (ข) ส่วนตัวแปร avail เป็นตัวแปรสำหรับเก็บค่าตำแหน่งอาร์เรย์ที่ว่างสำหรับจัดเก็บโหนดข้อมูลตัวใหม่ที่จะนำเพิ่มในทรี ในที่นี้ ตัวแปร avail จะเก็บค่า 12

10.5.3 การออกแบบการสร้างเอวีแอลทรี

ในหัวข้อนี้จะอธิบายตัวอย่างการออกแบบโครงสร้างเอวีแอลทรีด้วยลิงค์ลิสต์ โดยใช้ตัวอย่างคำสั่งภาษาซีในลักษณะการเขียนโปรแกรมเชิงวัตถุ มีรายละเอียดดังนี้

1) การออกแบบโครงสร้างโหนดในการจัดเก็บข้อมูลในเอวีแอลทรี ดังนี้

```
typedef int itemType;
typedef struct AvlNodeData{
    itemType  element;  // the data in the node
    struct AvlNodeData   *left;
    struct AvlNodeData   *right;
    int  height;       // Height of node
}AvlNode;
```

2) การออกแบบคลาสสำหรับจัดการโครงสร้างข้อมูลเอวีแอลทรี ด้วยการออกแบบคลาสสำหรับโครงสร้างทรี โดยกำหนดให้ตัวแปร root ชี้ไปยังโหนดรากของทรีที่ออกแบบไว้ข้างต้น และออกแบบการดำเนินการต่างๆ สำหรับจัดการข้อมูลในโครงสร้างทรี เช่น เมท็อด insertNewNode() สำหรับการเพิ่มโหนดข้อมูลในทรี เป็นต้น มีรายละเอียดคำสั่ง ดังนี้

```
class AvlTree{
 private:  AvlNode *root;
 public:
     AvlTree( ) { //Constructor
          root = NULL; //empty tree
     }

     void insertNewnode( itemType x );
```

```
private: //private methods
    AvlNode *createNewNode (itemType data);
    AvlNode *rotateRightChild(AvlNode *a );
    AvlNode *rotateLeftChild(AvlNode *a );
    AvlNode *doubleRotateRightChild(AvlNode *a );
    AvlNode *doubleRotateLeftChild(AvlNode *a );
    int MAX(int x, int y);
    int Height( AvlNode *p );
    AvlNode *insert (itemType x, AvlNode *t );
}
```

ต่อไปนี้เป็นตัวอย่างการออกแบบการทำงานของแต่ละการดำเนินการพื้นฐานของเอวีแอลทรี

1) createNewNode() เป็นฟังก์ชันสำหรับเพิ่มค่าข้อมูลใหม่เพื่อจัดเก็บในทรี มีคำสั่งภาษาซี ดังนี้

```
AvlNode *createNewNode (itemType data){
    AvlNode  *new_node;
    new_node =  new AvlNode;
    new_node ->element= data;
    new_node->left=new_node ->right = NULL;
    new_node ->height = 0;
    return(new_node);
}
```

สร้างโหนดใหม่ เพื่อจัดเก็บข้อมูล data โดยจะกำหนดตัวชี้โหนดซ้าย (left) และขวา (right) เป็นค่า NULL และกำหนดค่าความสูงของโหนดเป็น 0

2) rotateRightChild ( ) เป็นฟังก์ชันสำหรับการหมุนครั้งเดียวแบบหมุนโหนดลูกทางขวาไปซ้าย มีคำสั่งดังภาษาซี ดังนี้

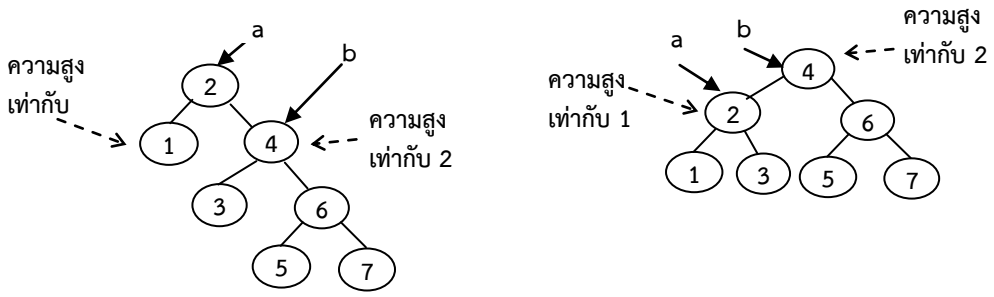
```
AvlNode *rotateRightChild(AvlNode *a ) {
    AvlNode *b = a->right;
    a->right = b->left;
    b->left = a;
    a->height = MAX( Height( a->left ), Height( a->right ) ) + 1;
    b->height = MAX( Height( b->right ), a->height ) + 1;
    return b;
}
```

การหมุนโหนดลูกทางขวา (โหนด b) ไปแทนที่โหนด a

รูปที่ 10.28 แสดงตัวอย่างในกรณีที่โหนดข้อมูล 2 มีความสูงของทรีย่อยทางขวา (ความสูงของโหนด 4) ต่างจากค่าความสูงของทรีย่อยทางซ้าย (ความสูงของโหนด 1) เท่ากับ 2 จะมีผล ทำให้ต้องหมุนโหนดลูกทางขวาไปแทนที่โหนด 1

กำหนดให้ a ชี้ที่โหนด 1 แล้วเรียกใช้งานฟังก์ชัน rotateRightChild(a) ภายในฟังก์ชันมีการทำงาน ดังนี้

1) ให้ b ชี้ที่โหนดลูกทางขวาของ a ด้วยคำสั่ง b = a->rightดังรูปที่ 10.28 (ก)
2) หมุนตำแหน่งโหนดb ไปแทนที่โหนด a ด้วยคำสั่ง a->right = b->left และ b->left = a จะได้ผลลัพธ์ดังรูปที่ 10.28 (ข)



(ก) ทรีที่ขาดคุณสมบัติความเป็นเอวีแอลทรีที่ตำแหน่ง a **(ข) ทรีหลังการหมุนโหนดขวาไปซ้าย**

รูปที่ 10.28 การหมุนจากโหนดขวาไปซ้าย

หลังจากหมุนโหนดลูกทางขวาแล้ว ดังรูปที่ 10.28 (ข) จะมีการกำหนดความสูงของทรีย่อย a และ b ใหม่ด้วยการพิจารณาจากค่าสูงสุดของความสูงระหว่างทรีย่อยทางขวาและทางซ้ายของแต่ละทรีย่อย ดังนี้

```
a->height = MAX( Height( a->left ), Height( a->right ) ) + 1;
b->height = MAX( Height( b->right ), a->height ) + 1;
```

1) การหาค่าสูงสุดจะเรียกใช้คำสั่ง MAX() ดังนี้

```
int MAX(int x, int y) {  return( (x) > (y) ? (x): (y)); }
```

2) การหาความสูงของทรีย่อย จะเรียกใช้งานฟังก์ชัน Height() ดังนี้

```
int Height( AvlNode *p ){
        if (p != NULL)  return(p-> height);
        elsereturn(-1);
}
```

3) rotateLeftChild() เป็นฟังก์ชันสำหรับการหมุนครั้งเดียวแบบหมุนโหนดลูกทางซ้ายไป เพื่อแก้ไขการขาดคุณสมบัติความเป็นเอวีแอลทรีของโหนด a ในกรณีของทรีย่อยทางซ้ายของโหนด a มีความสูงมากกว่าค่าความสูงของทรีย่อยทางขวาเท่ากับ 2 จึงต้องทำให้หมุนโหนดลูกทางซ้ายไปแทนที่โหนด a ดังนี้

```
AvlNode *rotateLeftChild(AvlNode *a ){
        AvlNode *b = a->left;
        a->left = b->right;
        b->right = a;
        a->height = MAX( Height( a->left ), Height( a->right ) ) + 1;
        b->height = MAX( Height( b->left ), a->height ) + 1;
        return b;
}
```

ให้ตัวแปร b ชี้ที่ทรีย่อยทางซ้ายของโหนด a หมุนโหนดลูกทางซ้ายไปแทนที่โหนด a

ปรับค่าความสูงของทรีย่อย a และ b ใหม่

3.5) doubleRotateRightChild() เป็นฟังก์ชันสำหรับการหมุนสองครั้งของโหนดลูกทางขวา ดังนี้

```
AvlNode *doubleRotateRightChild(AvlNode *a ){
        a->right = rotateLeftChild( a->right );
        return rotateRightChild( a);
}
```

ต้องทำการหมุนจากซ้ายไปขวาหนึ่งครั้ง และหมุนจากขวาไปซ้ายอีกหนึ่งครั้ง

3.6) doubleRotateLeftChild() เป็นฟังก์ชันสำหรับการหมุนสองครั้งของโหนดลูกทางซ้าย ดังนี้

```
AvlNode *doubleRotateLeftChild(AvlNode *a ) {
    a->left = rotateRightChild( a->left );
    return rotateLeftChild( a );
}
```

ต้องทำการหมุนจากขวาไปซ้ายหนึ่งครั้ง และหมุนจากซ้ายไปขวาอีกหนึ่งครั้ง

3.7) insertNewnode() เป็นฟังก์ชันสำหรับเพิ่มข้อมูลใหม่ในทรี ดังนี้

```
void insertNewnode( itemType x ) {
    root = insert( x, root );
}
```

ต้องเริ่มต้นค้นหาตำแหน่งที่ต้องการแทรกข้อมูล x ตั้งแต่โหนดรากด้วยคำสั่ง insert (x, root)

จากนั้นจึงหาตำแหน่งโหนดที่ต้องการเพิ่มโหนดใหม่โดยเรียกใช้ฟังก์ชัน insert() แบบเรียกวนทำซ้ำจนกว่าจะพบตำแหน่งโหนดที่ต้องการแทรกโหนดใหม่ โดยระหว่างการแทรกโหนดใหม่ จะมีการตรวจสอบว่าการแทรกโหนดใหม่ทำให้ทรีขาดคุณสมบัติความเป็นเอวีแอลทรีหรือไม่ หากใช่จะทำการหมุนทรีโดยจะเลือกหมุน 2 ครั้งหรือครั้งเดียวขึ้นอยู่กับลักษณะของตำแหน่งโหนดที่ทำให้ขาดความเป็นเอวีแอลทรีโดยมีรายละเอียดคำสั่งดังนี้

```
AvlNode *insert (itemType x, AvlNode *t )  {
    if  ( t == NULL )
       t = createNewNode (x);

    elseif (x< t->element  ) {
       t->left = insert( x, t->left );

       if (Height( t->left ) - Height( t->right ) == 2 )
          if ( x< t->left->element )
               t = rotateLeftChild(t );
          else   t = doubleRotateLeftChild (t);
    }
    else if (x> t->element  ) {
       t->right = insert( x, t->right);

       if (Height( t->right ) - Height( t->left ) == 2 )
          if (x > t->right->element )
               t = rotateRightChild (t);
          else t = doubleRotateRightChild (t);
    }
    t->height = MAX( Height( t->left ), Height( t->right ) ) + 1;
    return t;
}
```

หาก t เป็นค่า NULL ให้สร้างโหนดข้อมูล

หากข้อมูล x น้อยกว่า t->element ให้ไปแทรกโหนดใหม่ที่ลูกด้านซ้ายของ t

หลังแทรกโหนดใหม่ที่ทรีย่อยทางซ้ายของ t ให้ตรวจสอบความเป็นเอวีแอลทรีที่ความสูงของทรีย่อยทางซ้ายและทางขวาของ t หากความสูงมีค่าต่างกัน 2 ให้ทำการหมุนทรี

หากค่า x มากกว่า t->element ให้แทรกโหนดใหม่ที่ลูกด้านขวาของ t

เมื่อแทรกโหนดใหม่ที่ทรีย่อยทางขวาของ t เสร็จแล้ว จะตรวจสอบว่าต้องหมุนทรีหรือไม่

ปรับค่าความสูงของโหนด t ตามค่าสูงสุดของทรีย่อยทางซ้ายและทรีย่อยทางขวา

10.6 การค้นหาและการเรียงลำดับโดยใช้ไบนารีทรีเพื่อการค้นหา

เนื่องจากไบนารีทรีมีคุณสมบัติที่สำคัญอย่างหนึ่ง คือ โหนด x ใดๆ ค่าคีย์ทั้งหมดของทรีย่อยทางซ้ายจะมีค่าน้อยกว่าค่าคีย์ของโหนด x และค่าคีย์ทั้งหมดของทรีย่อยทางขวาจะมีค่ากว่าค่าคีย์ของโหนด x โดยลำดับขั้นตอนการค้นหาข้อมูลในทรีมีดังนี้

```
Algorithm find (k, v)
    if  v is null
        return null

    if k < key(v)
        return find(k,left (v))

    else if k = key(v)
         return v

    else  // k > key(v)
         return find(k,right (v))
```

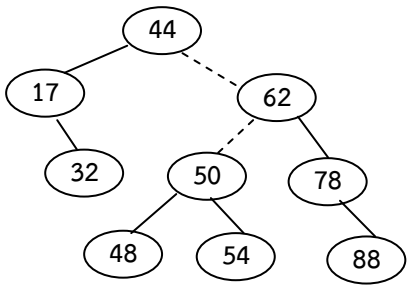
(1) กรณีโหนด v เป็นค่า null
แสดงว่าไม่พบค่าคีย์ที่ต้องการค้นหา

(2) กรณีค่าคีย์ (k) มีค่าน้อยกว่าค่าคีย์ของโหนด v
ให้ไปยังโหนดลูกทางซ้ายของ v

(3) กรณีค่าคีย์ (k) เท่ากับค่าคีย์ของโหนด v
ให้คืนค่าตำแหน่งโหนด v (พบโหนดที่ต้องการค้นหา)

(4) กรณีค่าคีย์ (k) มีค่ามากกว่าค่าคีย์ของโหนด v
ให้ไปยังโหนดลูกทางขวาของ v

จากรูปที่ 10.29 แสดงตัวอย่างลำดับโหนดตั้งแต่โหนดรากถึงโหนดที่มีค่าคีย์ที่ต้องการค้นหา ในกรณีค้นหาค่าคีย์ 50 ด้วยคำสั่ง Find (50, root) ดังนั้น k มีค่า 50และ v ชี้ที่โหนดราก และพบว่า 50มีค่ามากกว่าคีย์ของโหนด v (ค่า 44) จึงไปทางขวาของโหนด v ด้วยคำสั่ง find(k,right (v)) ดังนั้น v จึงชี้ที่โหนด 62จากนั้นพบว่า 50 มีค่าน้อยกว่าค่าคีย์ของโหนด v (ค่า 62) จึงต้องไปทางซ้ายของโหนด v ด้วยคำสั่ง find(k,left (v)) ซึ่งพบว่าเจอโหนดที่มีค่าคีย์เท่ากับ 50 โดยมีลำดับโหนดในทรีที่ถูกเข้าถึงเพื่อการค้นหาค่าคีย์ 50 คือ 44 62 และ 50ตามลำดับ ดังแสดงด้วยเส้นประในรูปที่ 10.29



รูปที่ 10.29 ตัวอย่างเส้นทาง (ตามเส้นประ) การค้นหาข้อมูล 50 จากตำแหน่งโหนดราก

จากขั้นตอนในการ ค้นหาค่าคีย์ในทรี พบว่าจำนวนครั้งในการเปรียบเทียบเพื่อค้นหาค่าคีย์จะไม่เกินกว่าค่าความสูงของทรี ดังนั้นไบนารีทรีเพื่อการค้นหาจึงเหมาะกับการใช้งานในลักษณะที่มีการค้นหาข้อมูลบ่อยๆ

ด้วยคุณสมบัติการเก็บค่าคีย์ของแต่ละโหนดในไบนารีทรีเพื่อการค้นหา หากเราใช้วิธีการเข้าถึงข้อมูลเพื่อแสดงค่าคีย์ของทุกโหนดในทรีด้วยวิธีการแวะผ่านทรีแบบอินออร์เดอร์ ตามคำสั่งต่อไปนี้

```
Algorithm printTree( ) {
if (root != NULL)
        inOrder( root );
 }
```

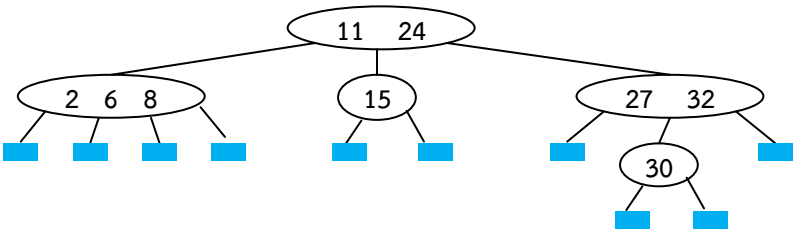
จากคำสั่งข้างต้นจะแสดงค่าข้อมูลแต่ละโหนดในทรีด้วยคำสั่งต่อไปนี้

```
Algorithm inOrder (p ) {
        if (left(p) != NULL )  inOrder(left(p));
        Print an element stored in node p
        if (right(p) != NULL)  inOrder(right(p));
}
```

จากรูปที่ 10.29 พบว่าเมื่อเข้าถึงข้อมูลทุกโหนดในทรีแบบอินออร์เดอร์ จะแสดงค่าข้อมูล 17 32 44 48 50 54 62 78 และ 88ตามลำดับ ซึ่งมีลักษณะแบบเรียงลำดับจากน้อยไปมาก ดังนั้น โครงสร้างไบนารีทรีจึงนิยมใช้ในการประยุกต์ใช้งานที่ต้องเข้าถึงข้อมูลแบบเรียงลำดับเป็นหลัก

นอกจากนี้ไบนารีเพื่อการค้นหาแล้ว สำหรับงานที่เน้นเรื่องการค้นหาข้อมูลและการเข้าถึงข้อมูลแบบเรียงลำดับบ่อยๆ ยังสามารถออกแบบการจัดเก็บข้อมูลในทรีแบบค้นหาได้หลายทาง ซึ่งเป็นโครงสร้างข้อมูลทรีที่เรียกว่า มัลติเวย์เสริชทรี (Multi-way Search Tree) เป็นทรีที่มีการเก็บหาคีย์แบบเรียงลำดับ มีคุณสมบัติของโหนดในทรีดังนี้

- แต่ละโหนดที่ไม่ใช่โหนดใบจะมีโหนดลูกอย่างน้อย 2 โหนด และเก็บค่าคีย์จำนวน d-1 ค่า เมื่อ d คือ จำนวนโหนดลูก ดังนั้นแต่ละโหนดจะประกอบด้วยโหนดลูก v_1 , v_2 ... , v_d และเก็บค่าคีย์ k_1, k_2 ... , k_{d-1} โดยที่
  - ค่าคีย์ทั้งหมดในทรีย่อยที่มี v_1 เป็นโหนดราก จะมีค่าคีย์น้อยกว่า k_1
  - ค่าคีย์ทั้งหมดในทรีย่อยที่มี v_i เป็นโหนดราก จะมีค่าคีย์อยู่ระหว่าง k_{i-1} และ k_i เมื่อ i=2, 3, ..., d-1
  - ค่าคีย์ทั้งหมดในทรีย่อยที่มี v_d เป็นโหนดราก จะมีค่าคีย์มากกว่า k_{d-1}
- โหนดใบจะไม่ได้เก็บข้อมูลของทรี เพื่อใช้เป็น placeholders



รูปที่ 10.30 ตัวอย่างโครงสร้างข้อมูลทรีแบบค้นหาได้หลายทาง

จากรูปที่ 10.30 แสดงตัวอย่างโครงสร้างข้อมูลทรี แบบค้นหาได้หลายทาง โดยมีโหนดรากที่เก็บค่าคีย์จำนวน 2 ค่า คือ คีย์แรกเท่ากับ 11 และคีย์ที่สอง เท่ากับ 24 และประกอบด้วยโหนดลูกจำนวน 3 โหนดดังนี้

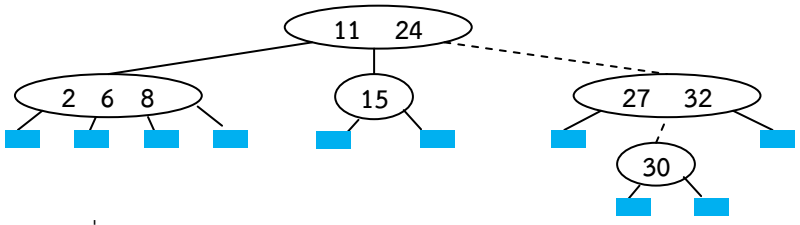
- โหนดลูกทางซ้ายสุดประกอบด้วยค่าคีย์ 2 6 และ 8 ซึ่งมีค่าน้อยกว่าค่าคีย์แรก (ค่า 11) ทั้งหมด
- โหนดลูกตรงกลาง ประกอบด้วยค่าคีย์ 15 ซึ่งเป็นค่าคีย์ที่มีค่ามากกว่าค่าคีย์แรก (ค่า 11) และน้อยกว่าค่าคีย์ที่สอง (ค่า 24)

- โหนดลูกทางขวาสุด ประกอบด้วยค่าคีย์ 27 และ 32 ซึ่งมีค่ามากกว่าค่าคีย์ที่สอง (ค่า 24) ทั้งหมดโดยโหนดลูกนี้มีโหนดลูกตรงกลางที่ประกอบด้วยค่าคีย์ 30 ซึ่งเป็นค่าที่อยู่ระหว่าง 27 และ 32

จากคุณสมบัติของมัลติเวย์เสริชทรีจะพบว่าหากเข้าถึงทุกค่าคีย์ภายในทรีแบบอินออร์เดอร์จะเป็นการเข้าถึงคีย์แบบเรียงลำดับจากน้อยไปมาก ตัวอย่างเช่น หากเข้าถึงทุกค่าคีย์ของทุกโหนดในรูปที่ 10.30 จะได้ผลลัพธ์ 2 6 8 11 15 24 27 30 และ 32 ตามลำดับ ซึ่งเป็นลักษณะการเข้าถึงข้อมูลแบบเรียงลำดับ นอกจากนี้การค้นหาค่าคีย์ในมัลติเวย์เสริชทรีจะใช้วิธีการคล้ายกับการค้นหาค่าคีย์ในไบนารีทรีเพื่อการค้นหา หากแต่ละโหนดที่ประกอบด้วยโหนดลูก $v_1, v_2, \ldots, v_d$ และเก็บค่าคีย์ $k_1, k_2 \ldots, k_{d-1}$ จะมีลำดับขั้นตอนในการค้นหาค่าคีย์ (k) ในทรีโดยเริ่มต้นที่โหนดราก ดังนี้

1) หากไม่ใช่โหนดใบ ทำการค้นหาค่าคีย์โดยการเปรียบเทียบทุกค่าคีย์ในโหนด
   - หากค่าคีย์k เท่ากับค่าคีย์ k_i ตัวใด ๆ ในโหนด (เมื่อ i=1, 2, ..., d-1)
     แสดงว่าพบค่าคีย์ที่ค้นหา
   - หากค่าคีย์k น้อยกว่าค่าคีย์ k_1
     ให้ไปค้นหาค่าคีย์ที่โหนดลูก v_1ต่อไป ด้วยการย้อนกลับไปทำข้อ 1
   - หากค่าคีย์ k อยู่ระหว่างค่าคีย์ k_{i-1} และค่าคีย์ k_i คู่ใด ($k_{i-1} < k < k_i$ เมื่อi=2, ..., d-1)
     ให้ไปค้นหาที่โหนดลูก v_iต่อไปด้วยการย้อนกลับไปทำข้อ 1
   - หากค่าคีย์ k มีค่าคีย์ k_{d-1}
     ให้ไปค้นหาที่โหนดลูก v_d ต่อไปด้วยการย้อนกลับไปทำข้อ 1
2) หากพบโหนดใบ แสดงว่าไม่พบคีย์ที่ต้องการค้นหา

จากรูปที่ 10.31 เมื่อต้องการค้นหาค่า 30 จะพบว่าค่าคีย์ 30 มีค่ามากกว่า 24 ของโหนดรากและไปเปรียบเทียบที่โหนดลูกทางขวาสุดของโหนดราก เนื่องจาก 30 มีค่าอยู่ระหว่าง 27 และ 32 ดังนั้นไปเปรียบเทียบค่าคีย์ต่อที่โหนดลูกตรงกลางของโหนดนี้ และพบตำแหน่งค่า 30 ที่ต้องการค้นหา รูปที่ 10.31 แสดงเส้นทางการเปรียบเทียบค่าคีย์เพื่อค้นหาค่า 30



รูปที่ 10.31 ตัวอย่างโครงสร้างข้อมูลทรีแบบค้นหาได้หลายทาง

10.7 บีทรี

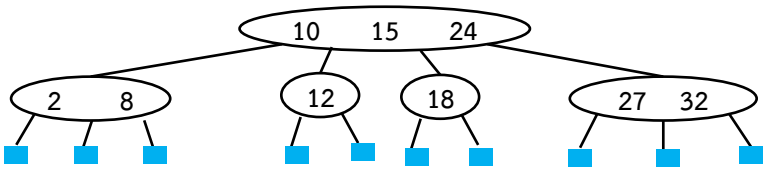
บีทรี (B-tree) เป็นทรีเพื่อการค้นหาที่มีโหนดลูกมากกว่า 2 โหนด หากเป็นบีทรีแบบออร์เดอร์ m จะเรียกว่าทรีแบบค้นหาได้ m ทาง (m-way Search Tree) ซึ่งเป็นทรีที่แต่ละโหนดที่ไม่ใช่โหนดใบ จะมีโหนดลูกอย่างน้อย ceil(m/2) โหนด และโหนดใบทั้งหมดในทรีต้องอยู่ในระดับเดียวกัน ซึ่งเป็นลักษณะทรีแบบบริบูรณ์ ตัวอย่างเช่น 2-3 ทรี (Two-three Tree)เป็นบีทรีที่สามารถค้นหาได้ 2 ถึง 3 เส้นทาง เป็นต้น ในหัวข้อนี้จะขอกล่าวถึงบีทรีที่สามารถค้นหาได้ 2-4 เส้นทาง เรียกว่า 2-4 ทรี (Two-four Tree)

(1) คุณสมบัติของ 2-4 ทรี

2-4 ทรี หรือ 2-3-4 ทรีหรือ (2,4) ทรีคือ บีทรีประเภทหนึ่งที่มีคุณสมบัติดังนี้

"โหนดที่ไม่ใช่โหนดใบจะมีโหนดลูกได้ไม่เกิน 4 โหนด และโหนดใบทั้งหมดจะอยู่ในระดับเดียวกัน"

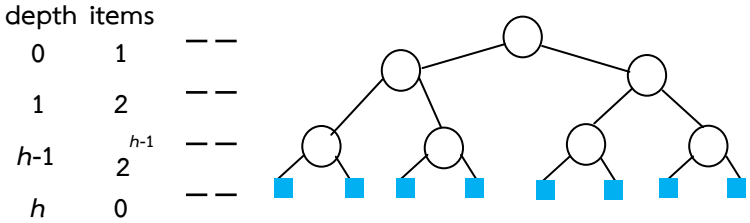
ใน 2-4 ทรี โหนดใดๆ ที่ไม่ใช่โหนดใบจะสามารถจัดเก็บค่าคีย์สูงสุดไม่เกิน 3 ค่า ซึ่งทำให้สามารถแบ่งประเภทของโหนดใน 2-4 ทรีได้ 3 ลักษณะขึ้นอยู่กับจำนวนโหนดลูกของโหนดนั้น โดยจะเรียกโหนดนั้นว่า 2-node 3-node หรือ 4-node ก็ต่อเมื่อโหนดนั้นมีการเชื่อมโยงไปยังโหนดลูกจำนวน 2 โหนด 3โหนด หรือ 4 โหนดตามลำดับ ดังรูปที่ 10.40แสดงตัวอย่าง 2-4 ทรี โดยโหนดรากประกอบด้วยค่าคีย์ 10 15 และ 24 ซึ่งเชื่อมโยงไปยังโหนดลูก 4 โหนด แสดงว่าโหนดรากนี้มีชนิดเป็น 4-node ส่วนโหนดลูกซ้ายสุดและโหนดลูกขวาสุดของโหนดรากจะเป็นชนิด 3-node เนื่องจากทั้งสองโหนดนี้ประกอบด้วย 2 คีย์ และเชื่อมโยงไปยังโหนดลูกได้ 3 เส้นทาง และโหนดลูกในตำแหน่งที่สองและตำแหน่งที่สามของโหนดรากจะเป็นชนิด 2-node เพราะมี 1 คีย์และสามารถเชื่อมโยงไปยังโหนดลูกได้จำนวน 2 โหนด



รูปที่ 10.40 ตัวอย่างโครงสร้างข้อมูลบีทรี

(2) ความสูงของ 2-4 ทรี

2-4 ทรีที่มีจำนวนสมาชิกในทรีเท่ากับ n จะมีค่าความสูงของทรีเท่ากับ O(log n) เนื่องจาก 2-4 ทรีจะมีความสูงของทรีสูงสุด เมื่อทุกโหนดในทรีเป็นแบบ 2-node ดังนั้นแต่ละระดับความลึกในทรีจะมีจำนวนโหนดอย่างน้อย 2^i โหนด เมื่อ i = 0, ... , h-1และไม่มีสมาชิกในระดับความลึก h ดังรูป 10.41



รูปที่ 10.41 ความสูงของโครงสร้างข้อมูลบีทรี

หาก h คือความสูงของทรี จะพบว่า $n >= 1 + 2 + 4 + ... + 2^{h-1} = 2^h - 1$, $h <= \log(n+1)$ดังนั้นเวลาในการค้นหาค่าคีย์ใน 2-4ทรีที่มีจำนวนสมาชิกเท่ากับ n ค่า จะใช้เวลาเท่ากับ O(log(n)) นั่นเอง

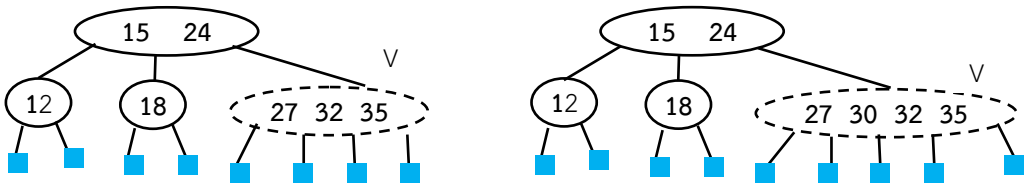
(3) การเพิ่มค่าคีย์ใน 2-4 ทรี

การเพิ่มค่าคีย์ใหม่ในทรี จะเริ่มต้นด้วยการค้นหาตำแหน่งโหนดที่ต้องการแทรกค่าคีย์ใหม่ (k) จากนั้นให้ใส่ค่าใหม่เพิ่มในโหนดนั้น ดังรูปที่ 10.42 แสดงตัวอย่างการเพิ่มค่าคีย์ใหม่ (30) ในทรี หาก v คือ ตำแหน่งโหนดที่ต้องการแทรกค่าคีย์ใหม่ ดังรูปที่ 10.42 (ก) อย่างไรก็ตามหลังการแทรกคีย์ใหม่

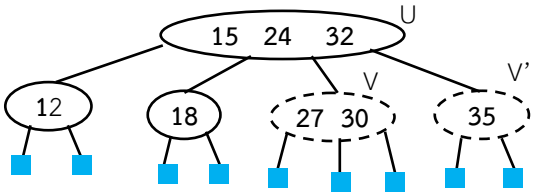
ในโหนด V หากโหนด V มีจำนวนคีย์มากกว่า 3 แสดงว่าเกิดปัญหาการล้นของโหนด ดังรูปที่ 10.42 (ข) ซึ่งพบว่าทำให้โหนด V มีจำนวนคีย์เท่ากับ 4 จึงเป็นโหนดแบบ 5-node

วิธีในการจัดการกรณีเกิดการล้นของค่าคีย์คือการแบ่งโหนดออกเป็น 2 โหนดดังนี้

1) หาก v1... v5 คือ โหนดลูกของโหนด Vและมีคีย์ k1 ... k4 ให้แบ่งโหนด V เป็น 2 โหนด เรียก V' และ V" ดังนี้
   1.1) V' เป็นบีทรีแบบ 3-โหนดที่เก็บค่าคีย์ k1 และ k2 และโหนดลูก v1 v2 และ v3
   1.2) V" เป็นชนิดแบบ 2-โหนดที่เก็บค่าคีย์ k4 และโหนดลูก v4 และ v5
2) หากโหนด Vมีโหนดพ่อ (เรียกโหนด U) ให้นำ k3 ขึ้นไปเก็บที่โหนด U
   กรณี V ไม่มีโหนดพ่อ (เมื่อ V เป็นโหนดรากของทรี) ให้ทำการสร้างโหนดใหม่เพื่อเก็บค่า k3
3) หากโหนด U เกิดเหตุการณ์ล้นของค่าคีย์ ให้จัดการการล้นด้วยการแบ่งโหนดเช่นเดียวกับโหนด V



(ก) ตำแหน่งโหนด V สำหรับเพิ่มคีย์ 30 (ข) หลังแทรกคีย์ 30 ในโหนด V



(ค) การแบ่งโหนด V เป็น V' และ V''

รูปที่ 10.42 ตัวอย่างการแทรกคีย์ใหม่ใน 2-4 ทรี

จากรูปที่ 10.42 (ข) โหนด V ประกอบด้วย 4 คีย์ คือ 27 30 32 และ 35 เมื่อแบ่งเป็น 2 โหนด คือ V' ประกอบด้วยคีย์ 27 และ 30 ส่วน V'' ประกอบด้วยคีย์ 35 ส่วนคีย์ 32 ถูกนำไปจัดเก็บในโหนดพ่อ ดังตัวอย่างรูปที่ 10.42 (ค)

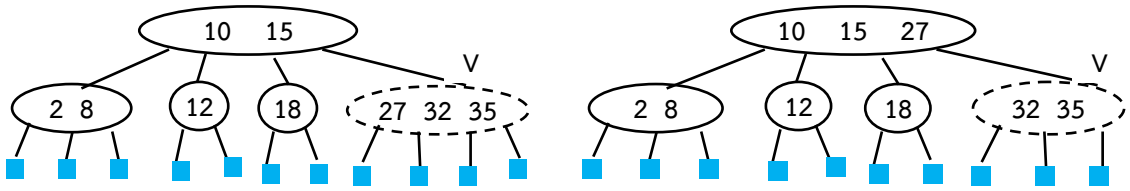
อัลกอริทึมในการเพิ่มค่าคีย์ใหม่ มีดังนี้

Algorithm *insertItem*(*k*, *o*)

1) ค้นหาตำแหน่งที่ต้องการเพิ่มโหนด Vในทรี
2) เพิ่มค่าคีย์ (k) ใหม่ ในโหนด V
3) ในขณะที่โหนด Vเกิดเหตุการณ์การล้นของโหนด (Overflow)
4) ถ้า Vเป็นโหนดราก
5) ให้สร้างโหนดใหม่แล้วกำหนดเป็นโหนดรากแทนโหนด V
6) แบ่งโหนด Vออกเป็น 2 โหนดด้วยการเรียกใช้ V← split(V)

(4) การลบค่าคีย์ใน 2-4 ทรี

การลบค่าข้อมูลใน 2-4 ทรี หากเป็นโหนดที่ติดกับโหนดใบให้ลบข้อมูลในโหนดนั้นหากไม่ใช่โหนดที่อยู่ติดกับโหนดใบให้นำข้อมูลที่เป็น inorder successor มาแทนที่ข้อมูลที่ต้องการลบ แล้วย้ายไปลบข้อมูลที่ถูกนำมาแทนที่แทน ดังตัวอย่างในรูปที่ 10.43 เมื่อต้องการลบค่าคีย์ 24 จะแทนค่าคีย์ 24 ด้วยค่าคีย์ 27 (เป็น inorder successor ของ 24) ของโหนด V แล้วไปดำเนินการลบค่าคีย์ 27 ในโหนดV แทน



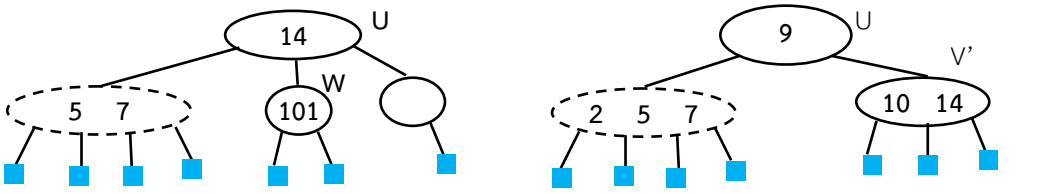
(ก) ตำแหน่งโหนด V ที่เก็บค่าคีย์ 27 **(ข) ย้ายคีย์ 27 ไปแทนที่คีย์ 24 ที่ถูกลบออกจากทรี**

รูปที่ 10.43 ตัวอย่างการลบคีย์ใน 2-4 ทรี

หลังการลบค่าคีย์ในทรี อาจทำให้เกิดเหตุการณ์ underflow คือ ทำให้โหนด V เหลือโหนดลูกเพียง 1 โหนด และไม่มีค่าคีย์ ซึ่งมีวิธีการจัดการปัญหากรณีโหนด V เกิด underflow เป็น 2 กรณี ดังนี้

(4.1) กรณีที่ 1 หากโหนดพี่น้องของ Vมีชนิดเป็น 2 โหนด คือ มีคีย์เดียว

กำหนดให้ W เป็นโหนดพี่น้องของ Vที่มีลักษณะเป็นแบบ 2 โหนด และ U เป็นโหนดพ่อของ V และ W ให้รวมโหนด V เก็บโหนดพี่น้อง W เรียกว่าโหนด V'พร้อมทั้งดึงค่าคีย์จากโหนด U ลงมารวมด้วย ซึ่งจะได้โหนด V' ที่เก็บค่าคีย์เรียงจากน้อยไปมาก และมีลักษณะเป็น 3-node ดังตัวอย่างในรูปที่ 10.44 และเนื่องจากจำนวนคีย์ของโหนด U ลดลง 1 ค่าจึงอาจมีผลทำให้โหนด U เกิดเหตุการณ์ underflow ได้ จึงอาจมีการวนทำซ้ำเพื่อจัดการโหนด U เช่นเดียวกับโหนด V



(ก) ตำแหน่งโหนด V ที่เกิด underflow (ข) การสร้างโหนด V' ที่รวมคีย์จากโหนด W และ U

รูปที่ 10.44 ตัวอย่างการรวมโหนดเพื่อแก้ปัญหา underflow ใน 2-4 ทรี

จากรูปที่ 10.44 เป็นตัวอย่างกรณีเกิด underflow ที่โหนด V และ โหนดพี่น้อง W มีคีย์เดียว ดังรูปที่ 10.44 (ก) จึงต้องรวมโหนด V และ W โดยการนำคีย์ 14 จากโหนด U มารวมกันเป็นโหนด V' ที่ประกอบด้วยค่าคีย์ 10 และ 14 ดังรูปที่ 10.44 (ข)

(4.2) กรณีที่ 2 หากโหนดพี่น้องมีชนิดเป็น 3-โหนด หรือแบบ 4-โหนด

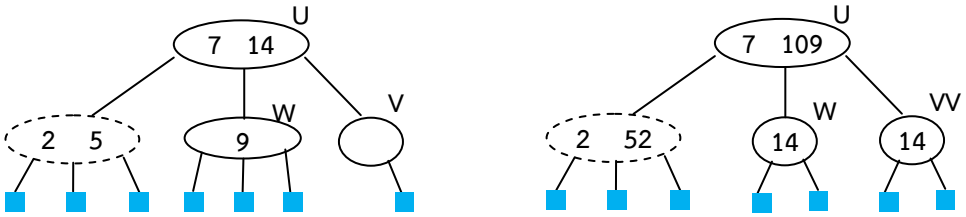
กำหนดให้ Wเป็นโหนดพี่น้องของ Vที่มีลักษณะเป็นแบบ 3 โหนด หรือ 4 โหนด ที่เก็บ 2 หรือ 3 คีย์ ตามลำดับ และ U เป็นโหนดพ่อของ V และ W มีขั้นตอนการทำงานในการ

แก้ปัญหา underflow ของโหนด V ให้ทำการโอนย้ายค่าคีย์จากโหนด W และโหนด U ดังนี้

- ย้ายค่าคีย์จากโหนด U ไปยังโหนด V
- และย้ายคีย์จากโหนด W ไปยังโหนด U

ในกรณีที่ 2 นี้จะไม่เกิดเหตุการณ์ underflow ที่โหนดใดๆขึ้นอีกอย่างแน่นอน เนื่องจากโหนดพี่น้องมีค่าคีย์ตั้งแต่ 2 ค่าขึ้นไปให้ใช้ และไม่ได้ลดจำนวนค่าคีย์ของโหนดพ่อ ต่างจากกรณีที่ 1

รูปที่ 10.45 เป็นตัวอย่างกรณีเกิด underflow ที่โหนด V และ โหนดพี่น้อง W มีคีย์มากกว่า 1 คีย์ ประกอบด้วย 9 และ 10 ดังรูปที่ 10.45 (ก) จึงสามารถแก้ปัญหา underflow ของโหนด V ได้ด้วยการโอนย้ายคีย์จากโหนดพี่น้อง (โหนด W) ของ V โดยการย้ายคีย์ 14 จากโหนด U ไปยังโหนด V และย้ายคีย์ 10 จากโหนด W ไปยังโหนด U ดังรูปที่ 10.44 (ข)



(ก) ตำแหน่งโหนด V ที่เกิด underflow (ข) โอนย้ายคีย์ระหว่างโหนด W U และ V

รูปที่ 10.45 ตัวอย่างการโอนย้ายคีย์เพื่อแก้ปัญหาunderflow ใน 2-4 ทรี

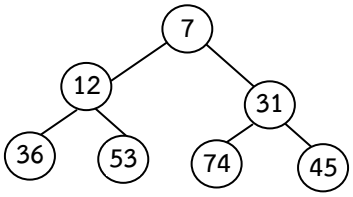
(5) ประสิทธิภาพในการดำเนินการสำหรับ 2-4 ทรี

หากทรีมีจำนวนโหนดเท่ากับ n โหนด และมีความสูง เท่ากับ log(n) จะพบว่าการดำเนินการในการเพิ่มค่าคีย์ในทรี จะใช้เวลาในการค้นหาตำแหน่งโหนดในการแทรกค่าคีย์ใหม่ไม่เกินความสูงของทรี และหากเกิดปัญหาการล้นของโหนดก็สามารถแก้ไขด้วยการแบ่งโหนดซึ่งอาจเกิดการวนทำซ้ำที่โหนดพ่อได้ แต่ก็จะใช้เวลาไม่เกินความสูงของทรีเช่นกัน ดังนั้นการดำเนินการเพิ่มคีย์ใหม่ในบีทรีจึงใช้เวลาไม่เกิน O(log n)

สำหรับการดำเนินการในการลบค่าคีย์ในทรี จะใช้เวลาในการหาตำแหน่งโหนดที่ต้องการลบและการแก้ปัญหาunderflowของโหนดด้วยการรวมคีย์หรือการโอนย้ายคีย์จากโหนดพี่น้อง ในเวลาไม่เกินความสูงของทรี หรือเท่ากับO(log n)เช่นกัน แม้ว่าจะเกิดเหตุการณ์ underflow ซ้ำที่โหนดพ่อก็ตาม

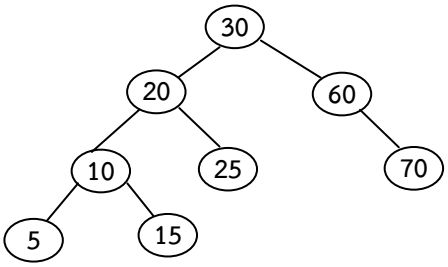
แบบฝึกหัด

1. จงแสดงขั้นตอนการสร้างทรีจากข้อมูลนำเข้า 30 10 15 25 7 8 9 ตามลำดับ
   ก) ขั้นตอนการสร้างโครงสร้างไบนารีทรีเพื่อการค้นหา
   ข) ขั้นตอนการสร้างโครงสร้างเอวีแอลทรี
2. จงเขียนอัลกอริทึมสำหรับเมท็อดต่อไปนี้ ในการจัดการกับข้อมูลในโครงสร้างไบนารีทรีเพื่อการค้นหา
   ก) FindMin() สำหรับการหาค่าโหนดที่มีข้อมูลต่ำสุด
   ข) FindMax() สำหรับการหาค่าโหนดที่มีข้อมูลสูงสุด
   ค) CountLeaf() สำหรับการนับจำนวนโหนดที่เป็นโหนดใบ
   ง) CountOneChild() สำหรับการนับจำนวนโหนดที่มีโหนดลูกทางซ้ายหรือโหนดลูกทางขวาด้านเดียว
3. จงแสดงขั้นตอนการสร้างโครงสร้างทรีแบบ Min-Heap จากข้อมูลนำเข้า 5 2 9 7 1ตามลำดับ
4. จงแสดงขั้นตอนการสร้างโครงสร้างทรีแบบ Max-Heap จากข้อมูลนำเข้า 5 2 9 7 1ตามลำดับ
5. จากรูปโครงสร้างข้อมูล Priority Queue ที่กำหนดให้ดังนี้

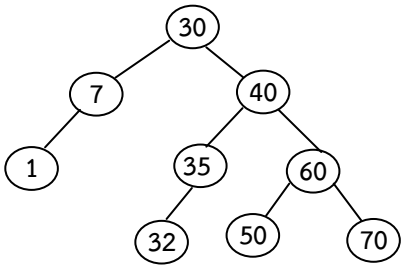


จงแสดงขั้นตอนการวาดรูปโครงสร้างทรีใหม่พร้อมทั้งแสดงขั้นตอนการเปลี่ยนแปลงค่าข้อมูลในอาร์เรย์เมื่อมีการเพิ่มข้อมูล 8

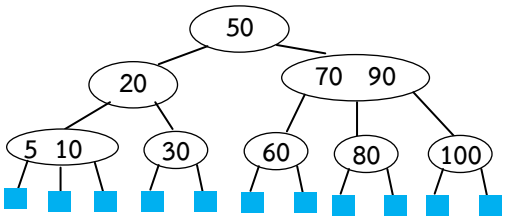
6. จากโครงสร้างเอวีแอลทรีที่กำหนดให้จงแสดงโครงสร้างเอวีแอลทรีที่ได้หลังการแทรกโหนด 90 และ 12 ตามลำดับ



7. จากโครงสร้างเอวีแอลทรีที่กำหนดให้จงแสดงโครงสร้างเอวีแอลทรีที่ได้หลังการลบโหนด 1 32 40 50 และ 60 (กำหนดให้การลบแต่ละครั้งให้เริ่มต้นที่ทรี)

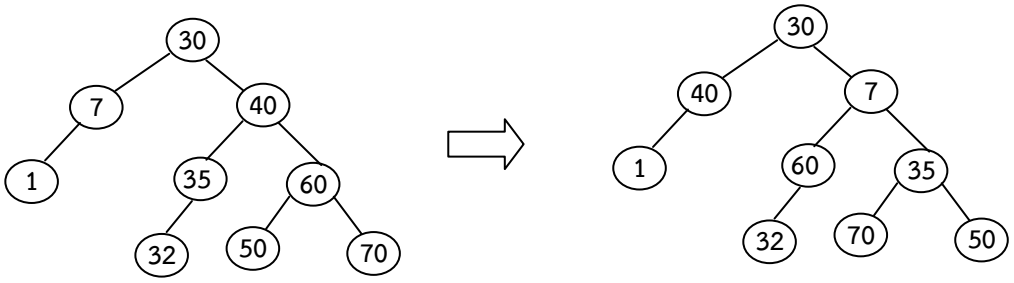


8. จากรูปโครงสร้าง 2-4 ทรีที่กำหนดให้



   1) จงแสดงโครงสร้างทรีที่ได้หลังการเพิ่มค่า 3 และ 15 ตามลำดับ
   2) จงแสดงโครงสร้างทรีที่ได้หลังการลบค่า 50

9. จงเขียนโปรแกรมเพื่อจัดเก็บข้อมูลเลขจำนวนเต็มในไบนารีทรี จำนวน n ตัว โดยให้มีเมท็อดในการดำเนินการกับทรีดังนี้
   ก) Insert() สำหรับเพิ่มโหนดในทรี
   ข) PreOrder() แสดงลำดับโหนดในทรีแบบ Preorder
   ค) InOrder() แสดงลำดับโหนดในทรีแบบ Inorder
   ง) PostOrder() แสดงลำดับโหนดในทรีแบบ Postorder
   จ) Swap() สำหรับสลับข้อมูลระหว่างโหนดทางซ้ายและโหนดทางขวา ดังรูปตัวอย่าง



10. จงเขียนโปรแกรมเพื่อจัดเก็บข้อมูลสตริงในโครงสร้างทรีแบบเอวีแอลทรี จำนวน n ตัว โดยกำหนดให้มีเมนูคำสั่งเพื่อดำเนินการกับข้อมูลในทรีดังนี้
   ก) Insert สำหรับเพิ่มโหนดในทรี
   ข) Delete สำหรับลบโหนดในทรี
   ค) LeftMost สำหรับค้นหาข้อมูลในโหนดซ้ายสุดของทรี โดยให้แสดงข้อมูลในทรีตามลำดับการค้นหาจากโหนดราก ถึงโหนดซ้ายสุดของทรี
   ง) RightMost สำหรับค้นหาข้อมูลในโหนดขวาสุดของทรี โดยให้แสดงข้อมูลในทรีตามลำดับการค้นหาจากโหนดราก ถึงโหนดขวาสุดของทรี
11. จงเขียนโปรแกรมเพื่อจัดเก็บข้อมูลนักศึกษาจำนวน n คน ในโครงสร้าง โดยแต่ละคนด้วยโครงสร้างข้อมูลแบบ 2-4 ทรี โดยข้อมูลของนักศึกษาประกอบด้วย รหัส ชื่อ นามสกุล และเกรดเฉลี่ย โดยให้โปรแกรมสามารถเพิ่มข้อมูล ค้นหาข้อมูลตามรหัสนักศึกษา และแสดงผลข้อมูลนักศึกษาทั้งหมดได้