หน่วยที่ 6

โครงสร้างข้อมูลแบบต้นไม้

หัวข้อเรื่อง

1. โครงสร้างข้อมูลแบบต้นไม้
2. ลักษณะโครงสร้างข้อมูลแบบต้นไม้
3. ศัพท์ที่ใช้เรียกแต่ละส่วนของต้นไม้
4. ไบนารีทรี
5. ลักษณะโครงสร้างข้อมูลแบบไบนารีทรี
6. การท่องเข้าไปในไบนารีทรี
7. ไบนารีเสิร์ชทรี

สาระสำคัญ

โครงสร้างข้อมูลแบบต้นไม้ (Tree Structure) เป็นโครงสร้างแบบไม่เชิงเส้น (Non Linear) การจัดเก็บข้อมูลมีลักษณะเป็นลำดับชั้น (Hierarchical) ข้อมูลแต่ละรายการจะเรียกว่าโหนด แต่ละโหนดจะมีพอยน์เตอร์ชี้ไปยังโหนดถัดลงไปเรื่อยๆ เป็นลำดับชั้น โดยโหนดเริ่มต้นจะเรียกว่า ราก หรือ รูทโหนด (Root Node) และจากรูทโหนดก็จะมีการเชื่อมโยงไปยังโหนดอื่นๆ ด้วยเส้นเชื่อมโยงที่เรียกว่า บรานซ์ (Branch)

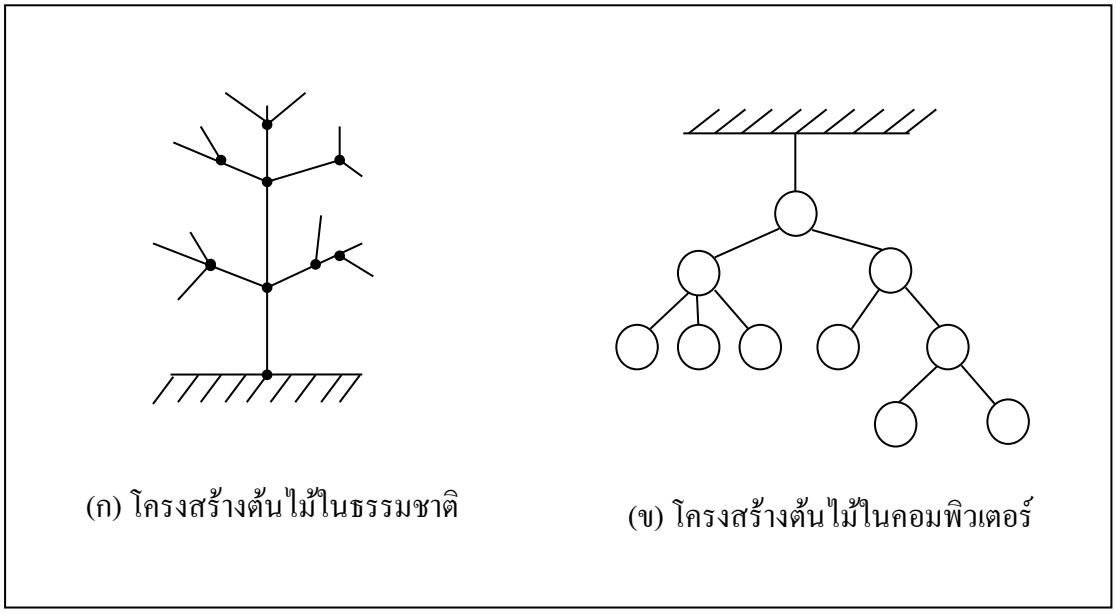
จุดประสงค์เชิงพฤติกรรม

1. อธิบายลักษณะโครงสร้างข้อมูลแบบต้นไม้ได้
2. อธิบายความหมายของศัพท์ที่ใช้เรียกแต่ละส่วนของต้นไม้ได้
3. อธิบายลักษณะของไบนารีทรีได้
4. อธิบายลักษณะโครงสร้างข้อมูลแบบไบนารีทรีได้
5. อธิบายการท่องเข้าไปในไบนารีทรีด้วยวิธีการต่างๆ ได้
6. อธิบายไบนารีเสิร์ชทรีได้

จากโครงสร้างข้อมูลที่กล่าวมาแล้วไม่ว่าจะเป็น อาร์เรย์ สแตก คิว หรือลิงค์ลิสต์ ล้วนแต่เป็นโครงสร้างข้อมูลแบบเชิงเส้น (Linear List) ทั้งสิ้น นั่นคือ มีลักษณะการเก็บข้อมูลแบบเรียงลำดับ แต่ถ้าเราต้องการเก็บข้อมูลที่มีลักษณะเป็นลำดับชั้น (Hierarchical) เช่นเก็บรายชื่อคณะผู้บริหารในบริษัทที่มีการไล่ระดับลงมาเรื่อยๆ หรือเก็บรายชื่อบรรพบุรุษ-ลูกหลานคนในตระกูล โครงสร้างข้อมูลที่จะเก็บข้อมูลในลักษณะนี้ ควรจะมีลักษณะเป็นลำดับชั้นเช่นกัน นั่นก็คือ จะใช้โครงสร้างข้อมูลแบบต้นไม้

โครงสร้างข้อมูลแบบต้นไม้

โครงสร้างข้อมูลแบบต้นไม้ เรียกอีกอย่างหนึ่งว่า โครงสร้างทรี (Tree Structure) เป็นโครงสร้างข้อมูลแบบไม่เชิงเส้นหรือไม่เรียงลำดับ (Non Linear) การจัดเก็บข้อมูลมีลักษณะเป็นลำดับชั้น (Hierarchical) เหมือนต้นไม้ในธรรมชาติ แต่จะแตกต่างกันตรงที่ต้นไม้ในธรรมชาติจะยืนต้นขึ้นสูงมีรากชี้ลงข้างล่าง แต่โครงสร้างต้นไม้ในคอมพิวเตอร์จะกลับตรงข้าม นั่นคือจะมีใบแตกกิ่งก้านสาขาลงด้านล่าง ส่วนด้านบนจะเป็นรากของต้นไม้

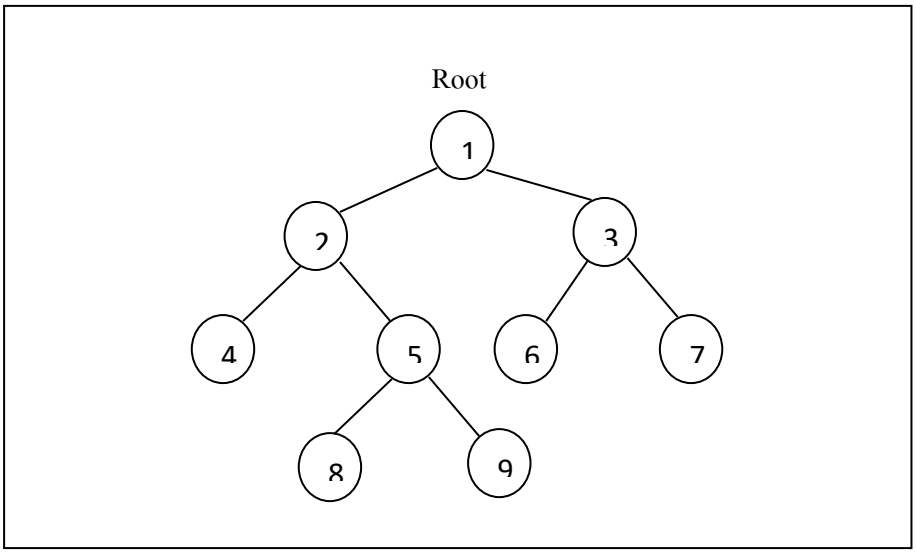


(ก) โครงสร้างต้นไม้ในธรรมชาติ (ข) โครงสร้างต้นไม้ในคอมพิวเตอร์

รูปที่ 6.1 เปรียบเทียบโครงสร้างต้นไม้ในธรรมชาติกับโครงสร้างต้นไม้ในคอมพิวเตอร์

ลักษณะโครงสร้างข้อมูลแบบต้นไม้

ลักษณะโครงสร้างข้อมูลแบบต้นไม้ ประกอบด้วย โหนด (Node) ซึ่งใช้เป็นที่เก็บข้อมูล โดยโหนดเริ่มต้นจะเรียกว่า ราก หรือรูทโหนด (Root Node) จากรูทโหนดจะมีการเชื่อมไปยังโหนดอื่นด้วยพอยน์เตอร์ และแต่ละโหนดก็จะมีพอยน์เตอร์ชี้ไปยังโหนดถัดกันไปเรื่อยๆ เป็นลำดับชั้นเหมือนต้นไม้จนถึงปลายโหนด ซึ่งเป็นโหนดที่ไม่มีการชี้ไปยังโหนดอื่นอีก นั่นคือพอยน์เตอร์ของโหนดที่ปลายจะเป็น NULL



รูปที่ 6.2 ลักษณะโครงสร้างข้อมูลแบบต้นไม้

ศัพท์ที่ใช้เรียกแต่ละส่วนของต้นไม้

เนื่องจากโครงสร้างข้อมูลแบบต้นไม้มีลำดับชั้นคล้ายกับการลำดับบรรพบุรุษ ดังนั้นจึงมีศัพท์ที่ใช้ในการเรียกแต่ละส่วนของต้นไม้เลียนแบบความสัมพันธ์ระหว่างพ่อลูก ดังต่อไปนี้

1. **รูทโหนด** (Root Node) คือ โหนดแรกสุดของโครงสร้างต้นไม้

2. **ระดับ** (Level) คือ ชั้นของโหนดซึ่งถูกระบุด้วยหมายเลข โดยที่จะกำหนดให้รูทโหนดเป็นระดับศูนย์ และไล่ระดับของโหนดด้วยการเพิ่มจำนวนทีละ 1

3. **บรานซ์** (Branch) เป็นเส้นเชื่อมโยงระหว่างโหนดที่มีความสัมพันธ์กัน ทำให้ทราบว่าเป็นโหนดพ่อแม่หรือโหนดลูก บางครั้งอาจเรียกว่า edge หรือ Link

4. **โหนดพ่อแม่** (Parent Node) คือ โหนดที่อยู่ระดับสูงกว่าโหนดที่ระบุอยู่หนึ่งระดับ โดยมีเส้นบรานซ์เชื่อมความสัมพันธ์ให้ทราบ

5. **โหนดลูก** (Child Node) คือ โหนดที่อยู่ต่ำกว่าโหนดที่ระบุอยู่หนึ่งระดับโดยมีเส้นบรานซ์เชื่อมความสัมพันธ์ให้ทราบ

6. **โหนดพี่น้อง** (Sibling Node) คือ โหนดที่อยู่ในระดับเดียวกัน และมีพ่อแม่เดียวกัน

7. **โหนดใบ** (Leaf Node) คือ โหนดที่ไม่มีโหนดถัดไปหรือโหนดที่ไม่มีลูกอยู่เลย

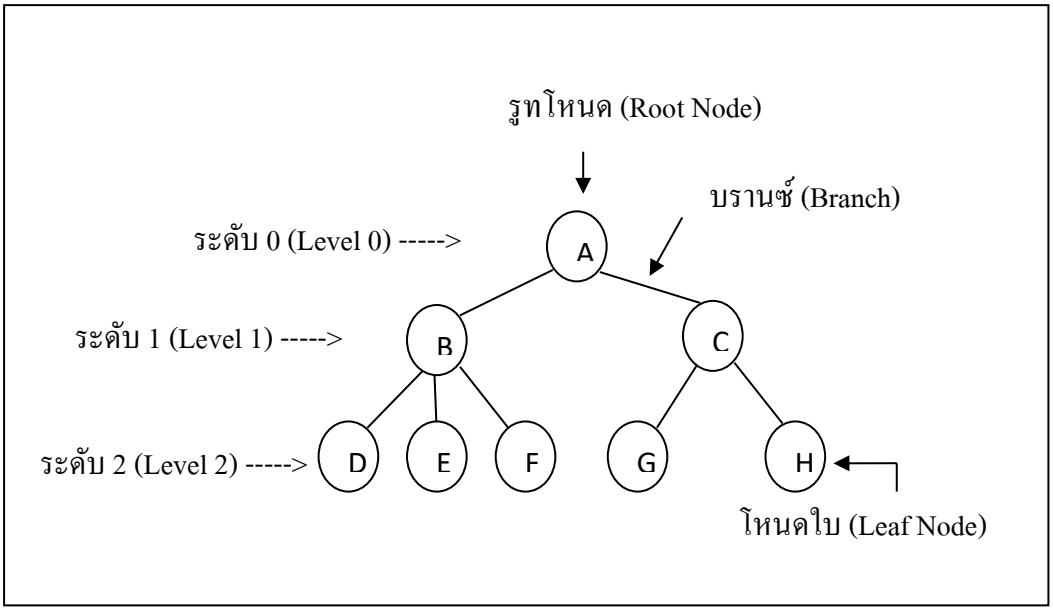
8. **โหนดบรรพชน** (Ancestor) คือ โหนดทุกโหนดในบรานซ์เดียวกัน โดยเริ่มลำดับตั้งแต่รูทโหนดมาจนถึงโหนดที่ระบุ

9. **โหนดสืบสกุล** (Descendant) คือ โหนดทุกโหนดที่อยู่บนต้นไม้ย่อยด้านซ้ายและต้นไม้ย่อยด้านขวาของโหนดที่ระบุ

10. **ดีกรี** (Degree) คือ จำนวนโหนดลูกทั้งหมดของโหนดที่ระบุ

11. **ต้นไม้ย่อย** (Subtree) คือ โครงสร้างแบบต้นไม้ในต้นไม้ทั้งต้น และโหนดแรกในต้นไม้ย่อยจะเป็นรูทโหนดของต้นไม้ย่อยนั้นๆ โดยทั่วไปจะแบ่งเป็นต้นไม้ย่อยด้านซ้าย (Left Subtree) และต้นไม้ย่อยด้านขวา (Right Subtree)

12. **ทรีว่าง** (Empty Tree) คือ โครงสร้างต้นไม้ที่ไม่มีโหนด นั่นคือจำนวนโหนดเท่ากับศูนย์



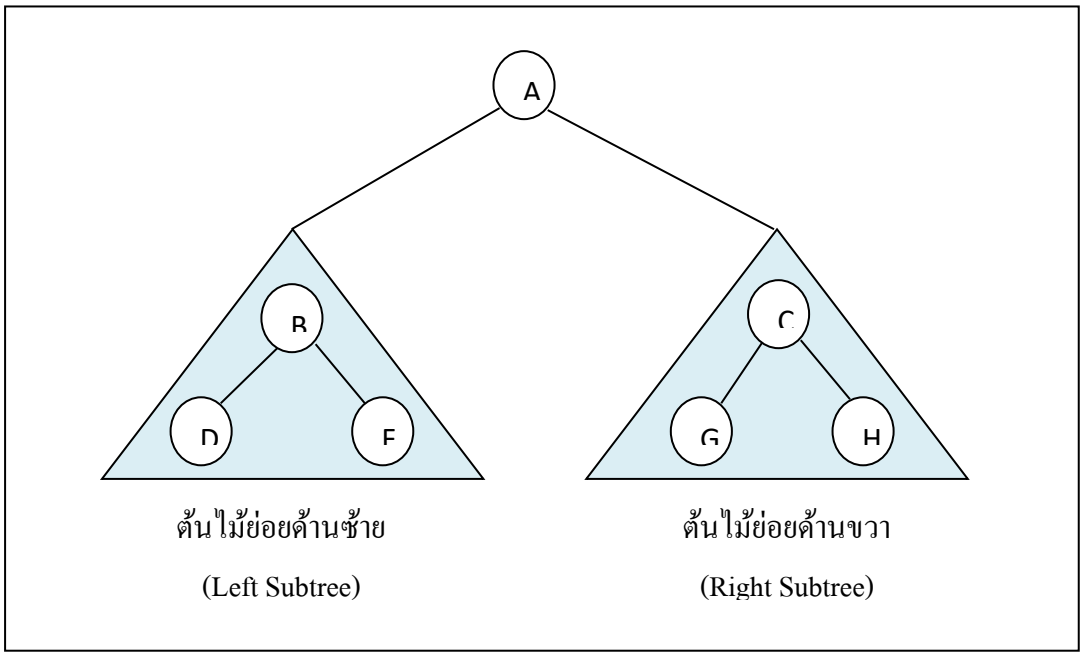
รูปที่ 6.3 โครงสร้างข้อมูลแบบต้นไม้และศัพท์ที่ใช้เรียก

จากรูปที่ 6.3 โครงสร้างข้อมูลแบบต้นไม้ มีรายละเอียดดังนี้

- รูทโหนด คือ A
- โหนดพ่อแม่ คือ A, B และ C
- โหนดลูกของ B คือ D, E และ F
- โหนดพี่น้อง คือ {B, C}, {D, E, F}, {G, H}
- โหนดใบ คือ D, E, F, G และ H ซึ่งก็คือโหนดที่ไม่มีลูก
- ดีกรีทั้งหมดของต้นไม้ คือ 7
- ดีกรีของ B คือ 3
- ต้นไม้ย่อย คือ B และ C

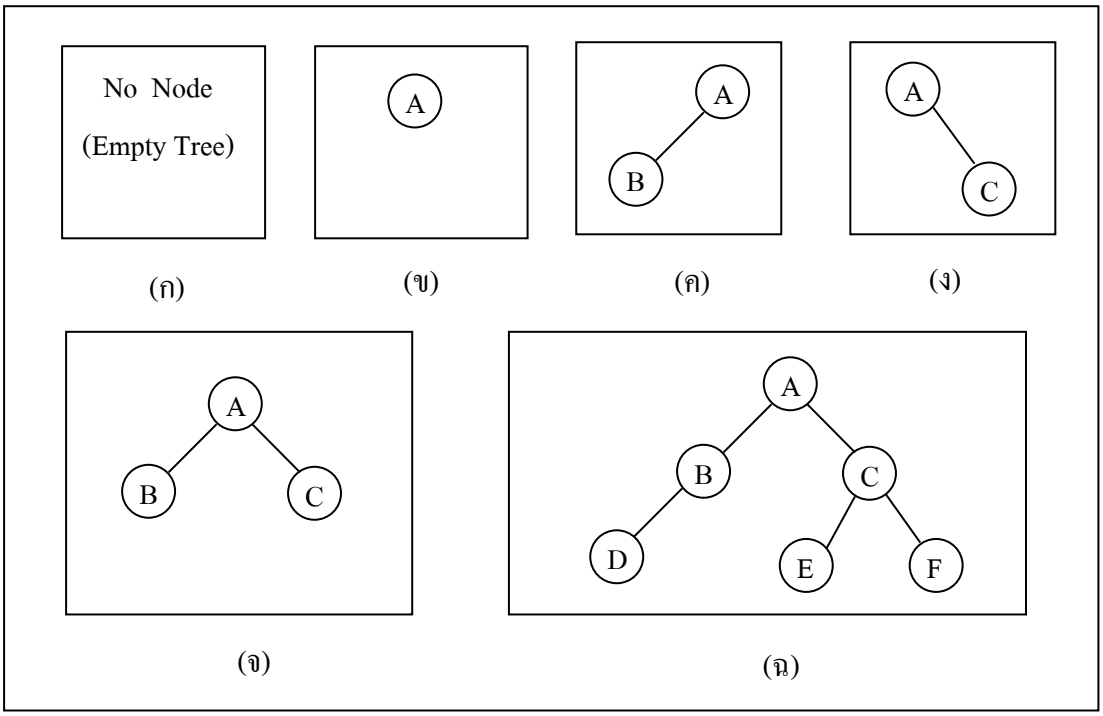
ไบนารีทรี

ไบนารีทรี (Binary Tree) เป็นโครงสร้างต้นไม้ที่มีรูปแบบเหมือนกับต้นไม้ทั่วไป โดยมีคุณสมบัติที่สำคัญ คือ โหนดแต่ละโหนดสามารถมี โหนดย่อย (Subtree) หรือ โหนดลูก (Child Node) ได้ไม่เกิน 2 โหนด โดยโหนดย่อยด้านซ้ายจะเรียกว่า ต้นไม้ย่อยด้านซ้าย (Left Subtree) และโหนดย่อยด้านขวาจะเรียกว่า ต้นไม้ย่อยด้านขวา (Right Subtree) ซึ่งต้นไม้ย่อยทั้งด้านซ้ายและด้านขวาต่างก็มีคุณสมบัติเป็นไบนารีทรี

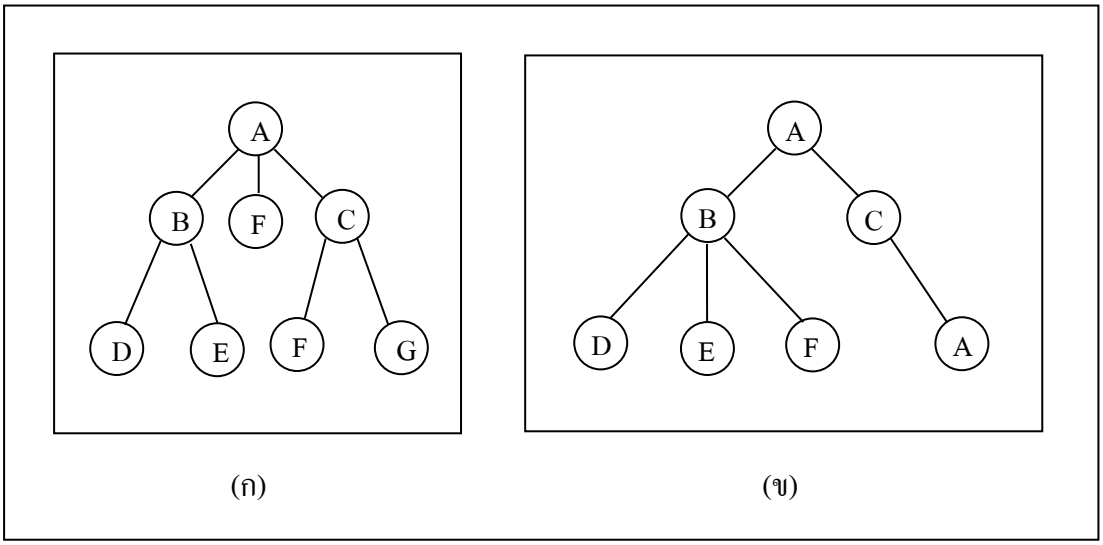


รูปที่ 6.4 ลักษณะโครงสร้างแบบไบนารีทรีที่มีต้นไม้ย่อยด้านซ้ายและด้านขวา

จากรูปที่ 6.4 แสดงโครงสร้างของไบนารีทรีที่มีทั้งต้นไม้ย่อยด้านซ้ายและต้นไม้ย่อยด้านขวา แต่โครงสร้างแบบไบนารีทรีไม่จำเป็นจะต้องมีโครงสร้างรูปแบบนี้เสมอไป บางครั้งอาจจะมีเฉพาะด้านซ้ายหรือด้านขวาหรือไม่มีเลย (Empty Tree) ก็ได้ ดังรูปที่ 6.5 แสดงลักษณะโครงสร้างไบนารีทรีรูปแบบต่างๆ ส่วนรูปที่ 6.6 เป็นโครงสร้างต้นไม้ที่ไม่ใช่ไบนารีทรี



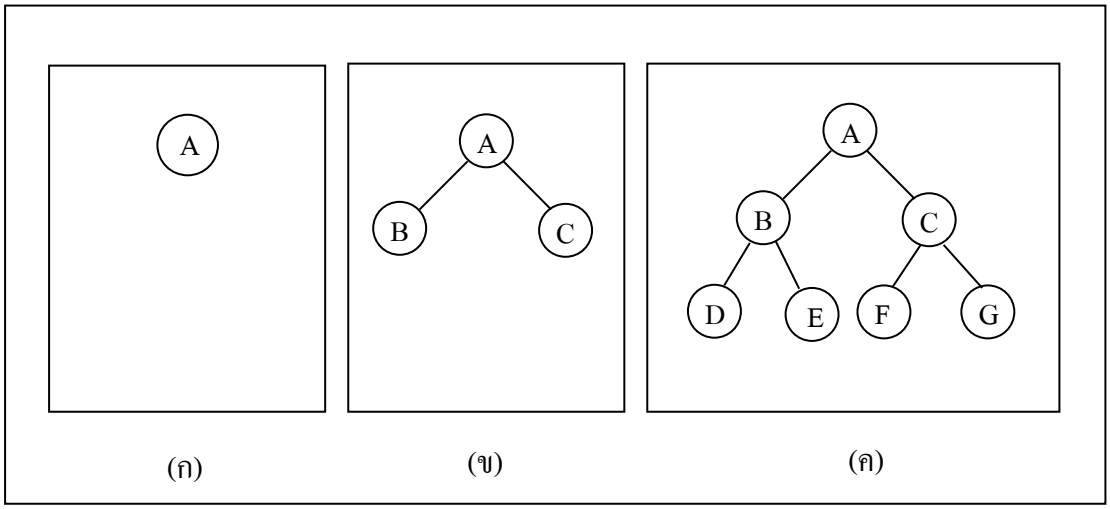
รูปที่ 6.5 ลักษณะโครงสร้างไบนารีทรีรูปแบบต่างๆ



รูปที่ 6.6 โครงสร้างต้นไม้ที่ไม่ใช่ไบนารีทรี

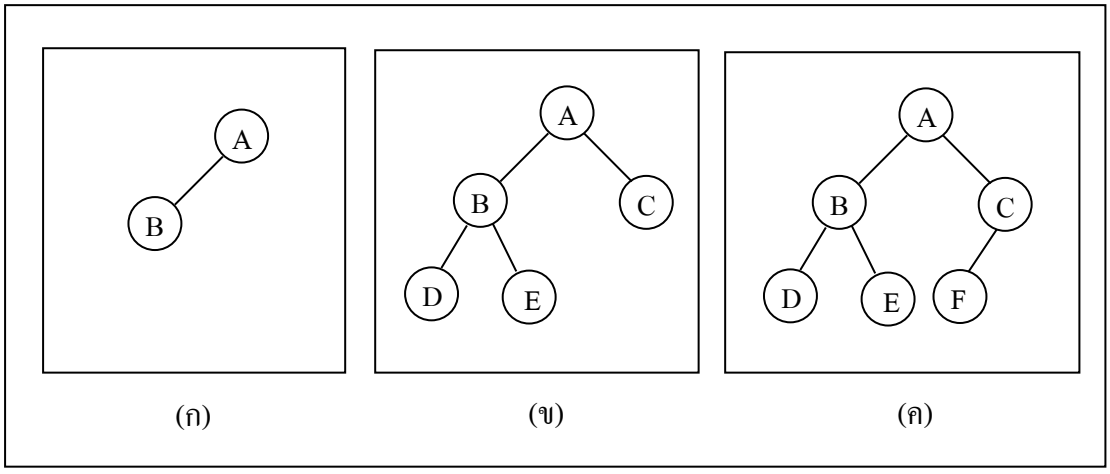
ไบนารีทรีแบบสมบูรณ์

ไบนารีทรีแบบสมบูรณ์ (Complete Binary Tree) จะมีลักษณะดังนี้คือ โหนดทุกโหนดจะต้องมีโหนดลูก 2 โหนดยกเว้นโหนดใบ และระดับของโหนดใบต้องอยู่ในระดับเดียวกัน ดังรูปที่ 6.7



รูปที่ 6.7 ลักษณะไบนารีทรีแบบสมบูรณ์

นอกจากนี้ยังมีไบนารีทรีที่เกือบสมบูรณ์ (Nearly Complete Binary Tree) นั่นคือเป็น ไบนารีทรีที่มีโหนดเต็มทุกโหนด ยกเว้นในระดับสุดท้ายที่มีโหนดไม่ครบตามที่กำหนด



รูปที่ 6.8 ลักษณะไบนารีทรีเกือบสมบูรณ์

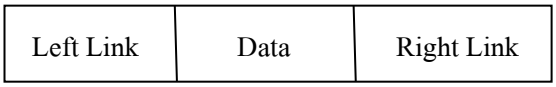
ลักษณะโครงสร้างข้อมูลแบบไบนารีทรี

ลักษณะโครงสร้างข้อมูลแบบไบนารีทรีเมื่อถูกนำไปใช้ในงานการเขียนโปรแกรม จะต้องสร้างตัวแปรขึ้นมาก่อน เพื่อแทนแต่ละโหนดที่อยู่ภายในไบนารีทรี โดยตัวแปรจะต้องประกอบด้วย 3 ส่วน ดังนี้

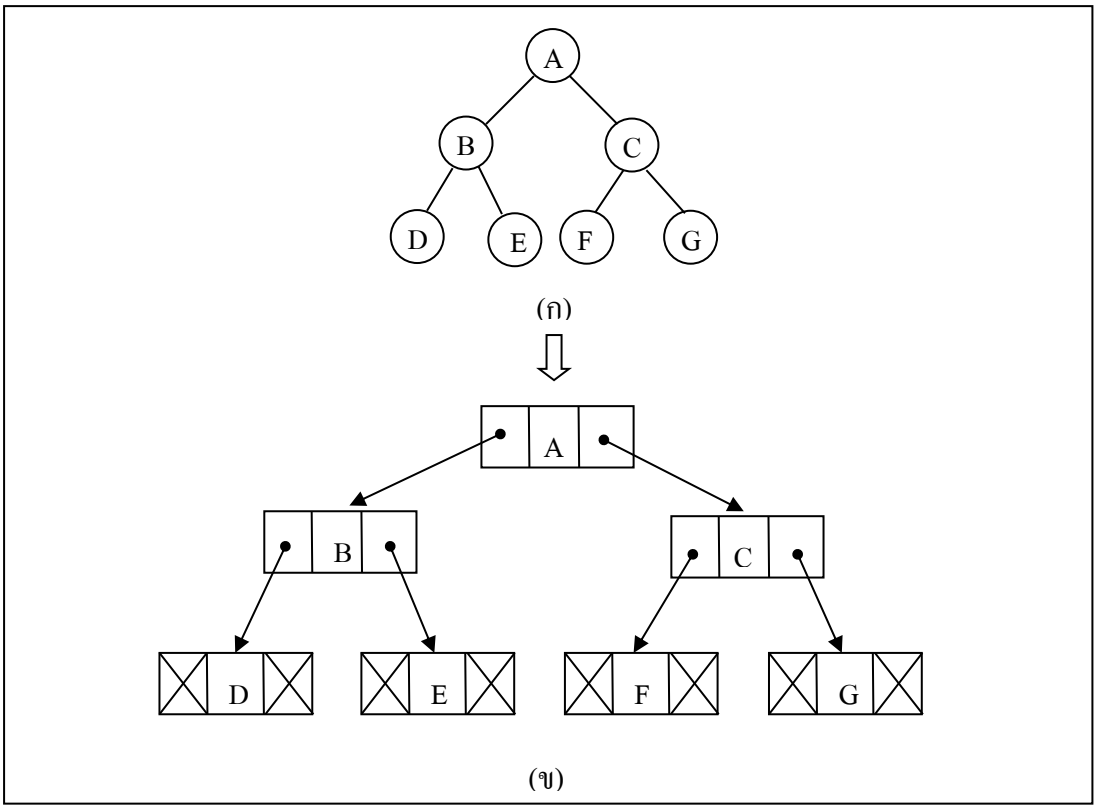
ส่วนที่ 1 Data Field คือ ส่วนที่ใช้เก็บข้อมูล จะเป็นตัวแปรชนิดใดขึ้นอยู่กับประเภทของข้อมูลที่ต้องการจัดเก็บในโหนด

ส่วนที่ 2 Left Link คือ พอยน์เตอร์ (Pointer) สำหรับชี้ไปยังต้นไม้ย่อยด้านซ้าย (Left Subtree) เพื่อที่จะบอกว่าโหนดถัดลงไปทางด้านซ้ายคือโหนดใด

ส่วนที่ 3 Right Link คือ พอยน์เตอร์ (Pointer) สำหรับชี้ไปยังต้นไม้ย่อยด้านขวา (Right Subtree) เพื่อที่จะบอกว่าโหนดถัดลงไปทางด้านขวาคือโหนดใด



รูปที่ 6.9 ลักษณะโครงสร้างข้อมูลแบบไบนารีทรี



รูปที่ 6.10 การสร้างไบนารีทรีจากลิงค์ลิสต์คู่

จากรูปที่ 6.10 เรามักจะสร้างโครงสร้างข้อมูลแบบไบนารีทรีขึ้นมาจากลิงค์ลิสต์คู่ จากรูปที่ 6.10 (ก) โครงสร้างแบบไบนารีเมื่ออยู่ในโปรแกรมคอมพิวเตอร์จะมีลักษณะดังรูปที่ 6.10 (ข)

การประกาศตัวแปรให้มีโครงสร้างข้อมูลแบบไบนารีทรีด้วยภาษา C++ ทำได้ดังนี้

```
struct  node
{
      node  * LLink;    /*  พอยน์เตอร์สำหรับชี้ต้นไม้ย่อยด้านซ้าย */
      int  data;          /*  สำหรับเก็บข้อมูล */
      node   *RLink;   /*  พอยน์เตอร์สำหรับชี้ต้นไม้ย่อยด้านขวา */
};
node    *Root;
```

การท่องเข้าไปในไบนารีทรี

การท่องเข้าไปในไบนารีทรี (Binary Tree Traversal) คือ การเดินเข้าไปในไบนารีทรีเพื่อที่จะเข้าไปดำเนินการกับข้อมูลที่เก็บไว้ในโหนดต่างๆ ซึ่งจะมีลักษณะเหมือนกับการเดินเยี่ยม (visit) โหนดต่างๆ เมื่อเยี่ยมครบทุกโหนดแล้วจึงเดินออกจากไบนารีทรี สำหรับวิธีการท่องเข้าไปในไบนารีทรีมีอยู่หลายวิธี แต่วิธีที่นิยมใช้มี 3 รูปแบบคือ

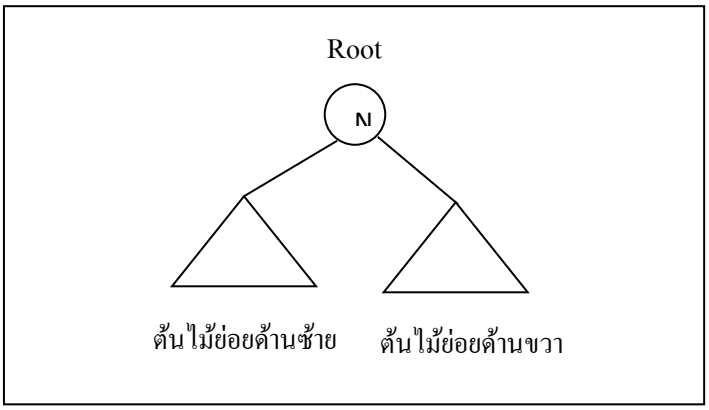
1. การท่องแบบพรีออเดอร์ (Preorder Traversal)
2. การท่องแบบอินออเดอร์ (Inorder Traversal)
3. การท่องแบบโพสต์ออเดอร์ (Postorder Traversal)

ในการอธิบายการท่องเข้าไปในไบนารีทรีแต่ละรูปแบบจะใช้คำย่อดังนี้

N แทน รูทโหนด (Root Node)

L แทน ต้นไม้ย่อยด้านซ้าย (Left Subtree)

R แทน ต้นไม้ย่อยด้านขวา (Right Subtree)

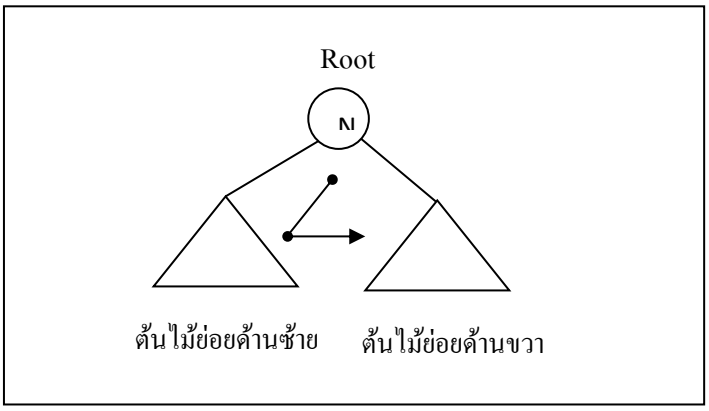


รูปที่ 6.11 สัญลักษณ์แทนค่าตำแหน่งของไบนารีทรี

การท่องแบบพรีออเดอร์ (Preorder Traversal)

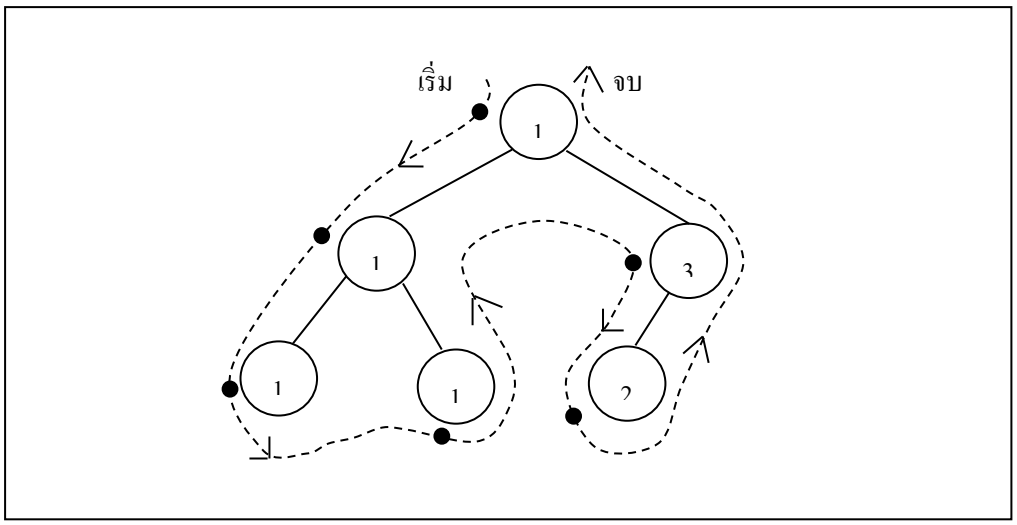
วิธีการท่องเข้าไปในไบนารีทรีแบบพรีออเดอร์นั้นจะใช้รูปแบบของ NLR ดังนี้

1. เข้าเยี่ยมรูทโหนด (N)
2. เดินเข้าเยี่ยมต้นไม้ย่อยด้านซ้าย (L)
3. เดินเข้าเยี่ยมต้นไม้ย่อยด้านขวา (R)



รูปที่ 6.12 แสดงการท่องไบนารีทรีแบบพรีออเดอร์

จากรูปที่ 6.12 วิธีการท่องเข้าไปในไบนารีทรีแบบพรีออเดอร์ จะเริ่มต้นจากจุดแรกคือรูทโหนด จากนั้นจึงเข้าไปเยี่ยมต้นไม้ย่อยด้านซ้าย และไปสิ้นสุดที่ต้นไม้ย่อยด้านขวา ซึ่งแสดงเส้นทางการท่องได้ดังรูปที่ 6.13 และสามารถเขียนฟังก์ชั่นแบบรีเคอร์ซีฟแสดงลำดับเส้นทางการท่องแบบ NLR ได้ดังโปรแกรมที่ 6.1



รูปที่ 6.13 แสดงเส้นทางการท่องเข้าไปในไบนารีทรีแบบพรีออเดอร์

จากรูปที่ 6.13 แสดงเส้นทางการท่องเข้าไปในไบนารีทรีแบบพรีออเดอร์ เส้นประแสดงถึงเส้นทางการเดินเข้าไปในไบนารีทรีว่ามีลำดับการผ่านแต่ละ โหนดอย่างไร จนกระทั่งเดินออกจากไบนารีทรี ส่วนจุดทึบบนเส้นประแสดงถึงการเข้าถึงโหนดซึ่งเราสามารถเข้าไปอ่านข้อมูล ลบข้อมูล แก้ไขข้อมูล หรือทำอะไรก็ได้กับข้อมูลที่เก็บอยู่ในโหนดนั้นเมื่อเดินมาถึงตำแหน่งนั้นๆ ส่วนหัวลูกศรที่อยู่บนเส้นประบอกถึงทิศทางในการท่องเข้าไปในไบนารีทรี และจากเส้นทางการท่องซึ่งถือว่าเป็นการเดินทางตามลำดับเส้นทางแบบ NLR ถ้าหากให้มีการพิมพ์ข้อมูลในแต่ละโหนดออกมาจะได้ข้อมูลดังนี้คือ 17, 13, 10, 15, 30 และ 25

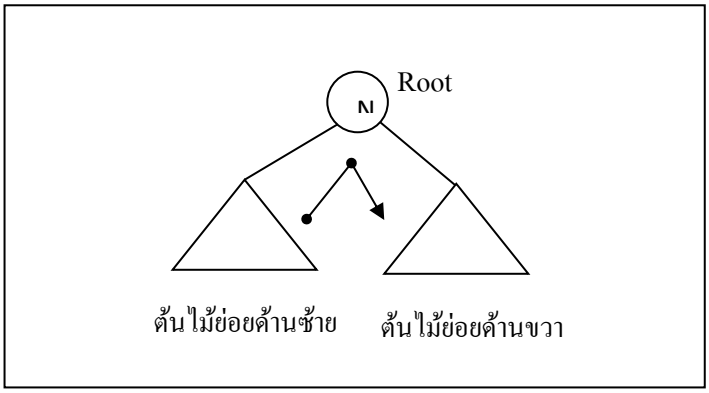
โปรแกรมที่ 6.1 การเขียนฟังก์ชั่นการท่องเข้าไปในไบนารีทรีแบบพรีออเดอร์

```
void  preorder(node  *p)
{
      if (p != NULL)
      {
            printf("%d\n",  p->data);    /* พิมพ์ผลลัพธ์ที่ p  */
            preorder(p->LLink);          /*  ท่อง L  แบบรีเคอร์ซีฟ    */
            preorder(p->RLink);          /*   ท่อง R  แบบรีเคอร์ซีฟ  */
    }
}
```

การท่องแบบอินออเดอร์ (Inorder Traversal)

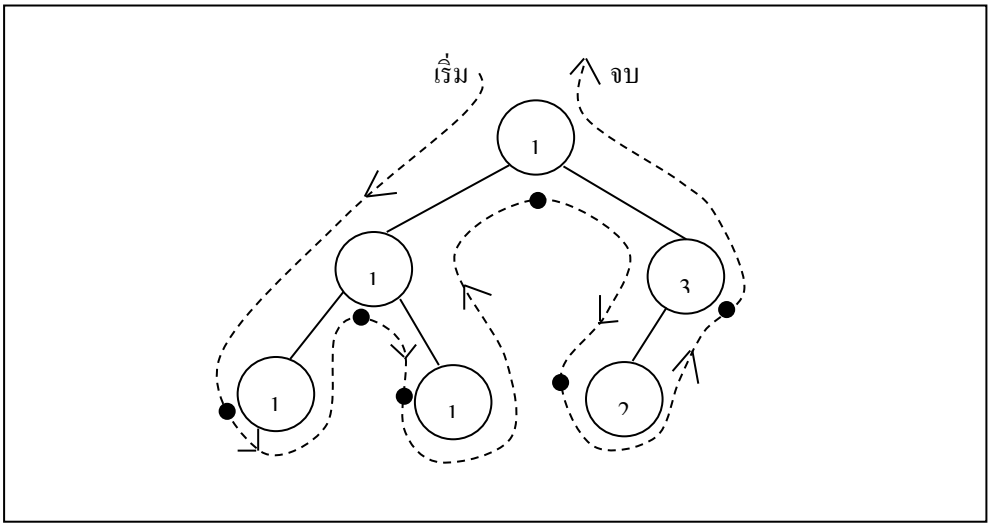
วิธีการท่องเข้าไปในไบนารีทรีแบบอินออเดอร์นั้นจะใช้รูปแบบของ LNR ดังนี้

1. เดินเข้าเยี่ยมต้นไม้ย่อยด้านซ้าย (L)
2. เข้าเยี่ยมรูทโหนด (N)
3. เดินเข้าเยี่ยมต้นไม้ย่อยด้านขวา (R)



รูปที่ 6.14 แสดงการท่องเข้าไปไบนารีทรีแบบอินออเดอร์

จากรูปที่ 6.14 วิธีการท่องเข้าไปในไบนารีทรีแบบอินออเดอร์ จะเริ่มที่การเข้าเยี่ยมต้นไม้ย่อยด้านซ้ายก่อนเป็นลำดับแรก จากนั้นจึงเข้าเยี่ยมรูทโหนด และไปสิ้นสุดที่ต้นไม้ย่อยด้านขวา ซึ่งแสดงเส้นทางการท่องได้ดังรูปที่ 6.15 และสามารถเขียนฟังก์ชั่นแบบรีเคอร์ซีฟแสดงลำดับเส้นทางการท่องแบบ LNR ได้ดังโปรแกรมที่ 6.2



รูปที่ 6.15 แสดงเส้นทางการท่องเข้าไปในไบนารีทรีแบบอินออเดอร์

จากรูปที่ 6.15 แสดงเส้นทางการท่องเข้าไปในไบนารีทรีแบบอินออเดอร์ ซึ่งเป็นการเดินทางตามลำดับเส้นทางแบบ LNR จะได้ข้อมูลที่เป็นผลลัพธ์จากการเข้าเยี่ยมโหนดแต่ละโหนด ดังนี้คือ 10, 13, 15, 17, 25 และ 30 หากพิจารณาข้อมูลผลลัพธ์จะเห็นได้ว่าข้อมูลที่ได้จะเรียงลำดับจากน้อยไปมาก

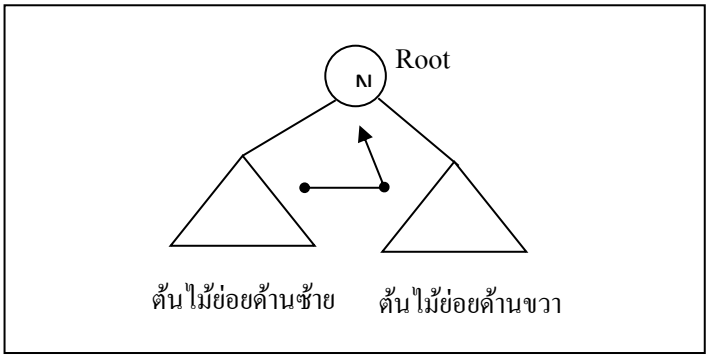
โปรแกรมที่ 6.2 การเขียนฟังก์ชั่นการท่องเข้าไปในไบนารีทรีแบบอินออเดอร์

```
void  inorder(node  *p)
{
      if (p != NULL)
      {
            inorder(p->LLink);          /*  ท่อง L  แบบรีเคอร์ซีฟ  */
            printf("%d\n",  p->data);   /* พิมพ์ผลลัพธ์ที่ p  */
           inorder(p->RLink);           /*   ท่อง R  แบบรีเคอร์ซีฟ  */
    }
}
```

การท่องแบบโพสต์ออเดอร์ (Postorder Traversal)

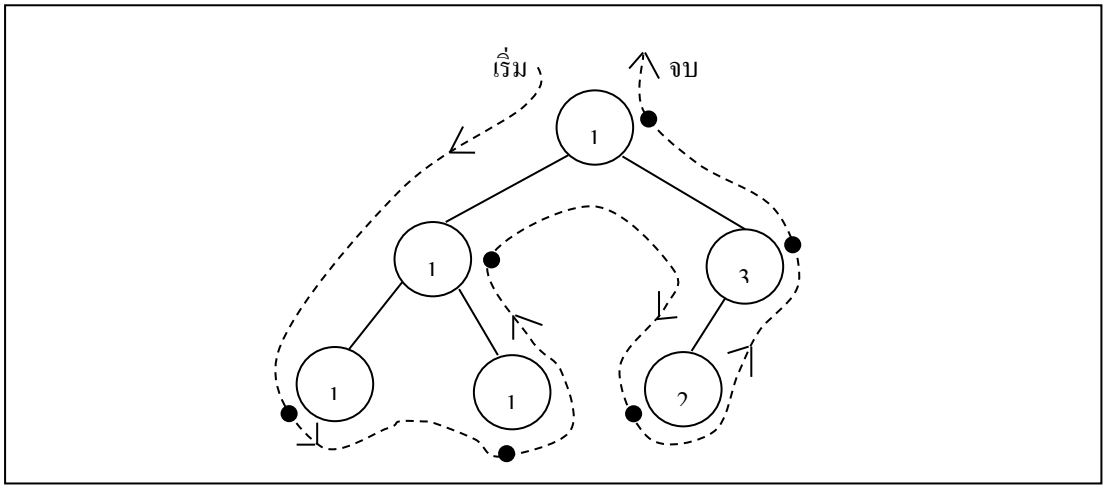
วิธีการท่องเข้าไปในไบนารีทรีแบบโพสต์ออเดอร์นั้นจะใช้รูปแบบของ LRN ดังนี้

1. เดินเข้าเยี่ยมต้นไม้ย่อยด้านซ้าย (L)
2. เดินเข้าเยี่ยมต้นไม้ย่อยด้านขวา (R)
3. เข้าเยี่ยมรูทโหนด (N)



รูปที่ 6.16 แสดงการท่องเข้าไปในไบนารีทรีแบบโพสต์ออเดอร์

จากรูปที่ 6.16 วิธีการท่องเข้าไปในไบนารีทรีแบบโพสต์ออเดอร์ จะเริ่มต้นเยี่ยมต้นไม้ย่อยด้านซ้ายก่อนเป็นลำดับแรก จากนั้นจึงเข้าไปเยี่ยมต้นไม้ย่อยด้านขวา และไปสิ้นสุดที่รูทโหนด ซึ่งแสดงเส้นทางการท่องได้ดังรูปที่ 6.17 และสามารถเขียนฟังก์ชั่นแบบรีเคอร์ซีฟแสดงลำดับเส้นทางการท่องแบบ LRN ได้ดังโปรแกรมที่ 6.3



รูปที่ 6.17 แสดงเส้นทางการท่องไบนารีทรีแบบโพสต์ออเดอร์

จากรูปที่ 6.17 แสดงเส้นทางการท่องเข้าไปในไบนารีทรีแบบโพสต์ออเดอร์ ซึ่งเป็นการเดินทางตามลำดับเส้นทางแบบ LRN จะได้ข้อมูลที่เป็นผลลัพธ์จากการเข้าเยี่ยมโหนดแต่ละโหนด ดังนี้คือ 10, 15, 13, 25, 30 และ 17

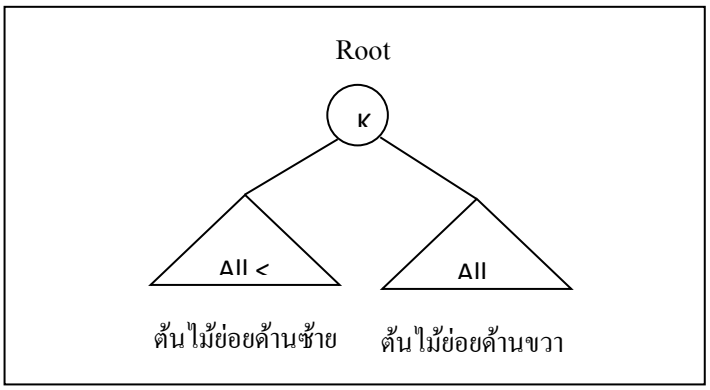
โปรแกรมที่ 6.3 การเขียนฟังก์ชั่นการท่องเข้าไปในไบนารีทรีแบบโพสต์ออเดอร์

```
void  postorder(node  *p)
{
      if (p != NULL)
      {
            postorder(p->LLink);         /*  ท่อง L แบบรีเคอร์ซีฟ  */
            postorder(p->RLink);        /*   ท่อง R แบบรีเคอร์ซีฟ  */
            printf("%d\n",  p->data);   /* พิมพ์ผลลัพธ์ที่ p   */
    }
}
```

ไบนารีเสิร์ชทรี

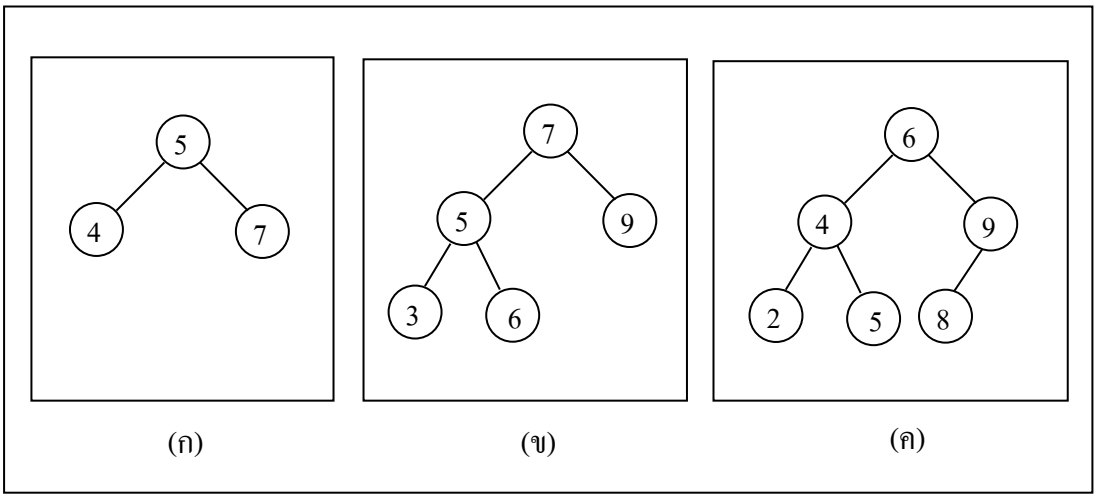
ไบนารีเสิร์ชทรี (Binary Search Tree : BST) เป็นการนำไบนารีทรีมาประยุกต์ใช้งานเพื่อการค้นหาข้อมูล ซึ่งไบนารีเสิร์ชทรีประกอบด้วยคุณสมบัติดังนี้

1. ทุกๆ โหนดที่อยู่ในต้นไม้ย่อยด้านซ้าย จะต้องมีค่าน้อยกว่ารูทโหนด
2. ทุกๆ โหนดที่อยู่ในต้นไม้ย่อยด้านขวา จะต้องมีค่ามากกว่าหรือเท่ากับรูทโหนด
3. ต้นไม้ย่อยทั้งสองด้านจะต้องมีคุณสมบัติของไบนารีเสิร์ชทรี (ตามข้อ 1 และข้อ 2)

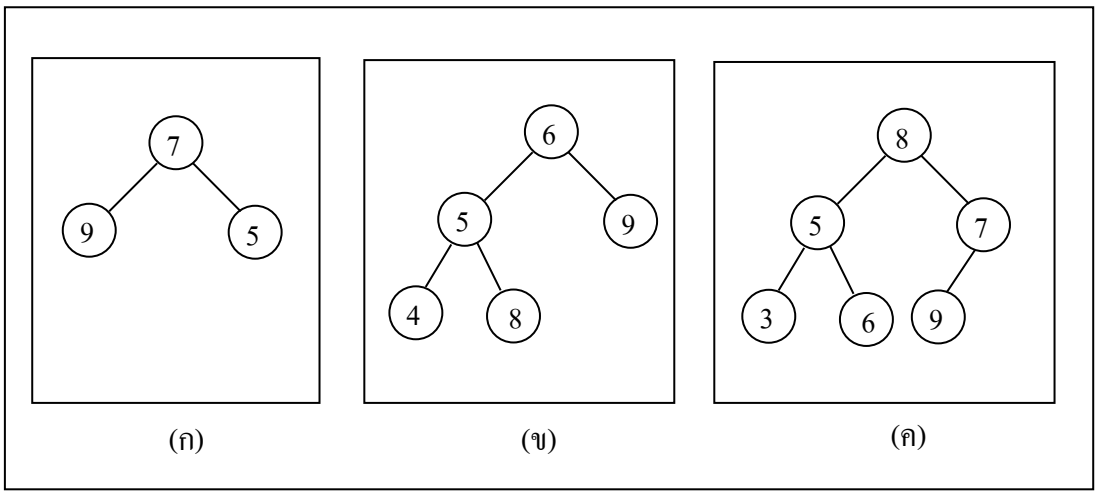


รูปที่ 6.18 คุณสมบัติของไบนารีเสิร์ชทรี

จากรูปที่ 6.18 คุณสมบัติของไบนารีเสิร์ชทรีค่าคีย์ที่บรรจุอยู่ในต้นไม้ย่อยด้านซ้ายจะมีค่าน้อยกว่ารูทโหนด และค่าคีย์ที่บรรจุอยู่ในต้นไม้ย่อยด้านขวาจะมีค่ามากกว่าหรือเท่ากับรูทโหนด



รูปที่ 6.19 ลักษณะไบนารีเสิร์ชทรีที่ถูกต้อง



รูปที่ 6.20 ไบนารีเสิร์ชทรีที่ไม่ถูกต้อง

การสร้างไบนารีเสิร์ชทรี

การสร้างไบนารีเสิร์ชทรี เพื่อความเข้าใจยิ่งขึ้นให้พิจารณาตัวอย่างที่ 6.1

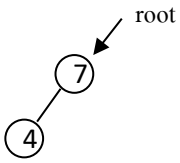
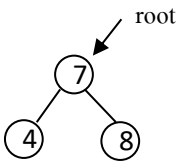
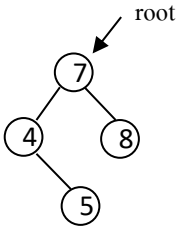
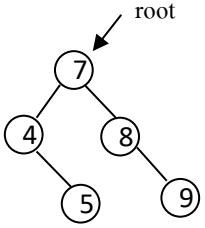
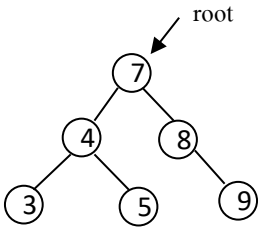
ตัวอย่างที่ 6.1 จงสร้างไบนารีเสิร์ชทรีจากข้อมูลที่กำหนดให้ต่อไปนี้

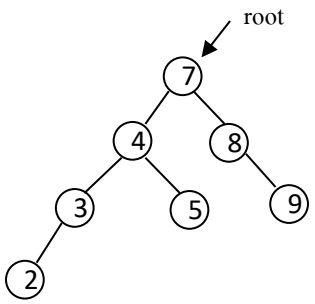
7 4 8 5 9 3 2

ในการสร้างไบนารีเสิร์ชทรีที่ชี้โดย root มีลำดับขั้นตอนแสดงดังตารางที่ 6.1

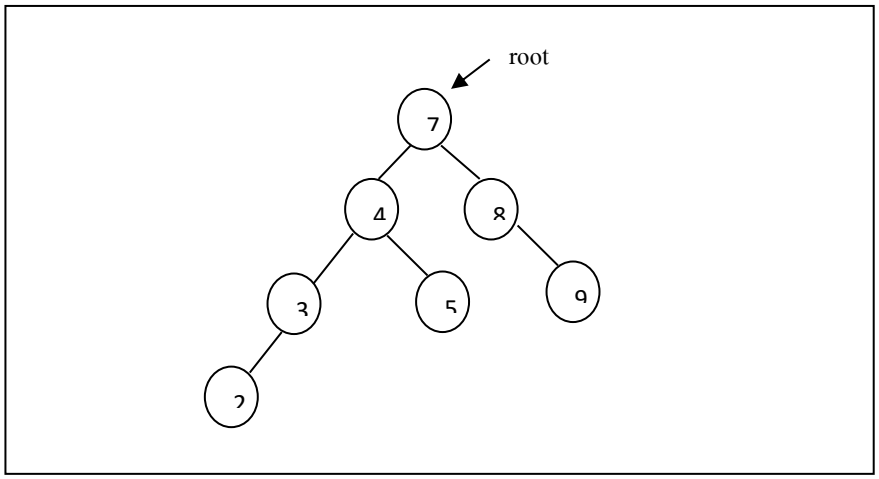
ตารางที่ 6.1 แสดงลำดับขั้นตอนการสร้างไบนารีเสิร์ชทรี

การดำเนินการ	ภาพของต้นไม้
1. เริ่มต้นกำหนดให้ root เป็น Null แสดงว่าเริ่มต้น ต้นไม้มีสภาพว่างเปล่า	● root
2. อ่านค่า 7 มาสร้างเป็นโหนด (7) แล้วนำไปแทรกในต้นไม้ แต่เนื่องจากต้นไม้ยังว่างเปล่าโหนดแรกที่สร้างคือ 7 จึงถือเป็นรูทโหนด	root ↙ (7)

การดำเนินการ	ภาพของต้นไม้
3. อ่านค่า 4 มาสร้างเป็นโหนด ④ แล้วนำไปแทรกในต้นไม้ โดยเทียบ 4 กับ 7 เนื่องจาก 4 มีค่าน้อยกว่า 7 จึงแทรกด้านซ้ายของ 7	
4. อ่านค่า 8 มาสร้างเป็นโหนด ⑧ แล้วนำไปแทรกในต้นไม้ โดยเทียบ 8 กับ 7 เนื่องจาก 8 มีค่ามากกว่า 7 จึงแทรกด้านขวาของ 7	
5. อ่านค่า 5 มาสร้างเป็นโหนด ⑤ แล้วนำไปแทรกในต้นไม้ โดยเทียบ 5 กับ 7 เนื่องจาก 5 มีค่าน้อยกว่า 7 จึงท่องลงมาทางซ้าย เทียบ 5 กับ 4 เนื่องจาก 5 มีค่ามากกว่า 4 จึงแทรกด้านขวาของ	
6. อ่านค่า 9 มาสร้างเป็นโหนด ⑨ แล้วนำไปแทรกในต้นไม้ โดยเทียบ 9 กับ 7 เนื่องจาก 9 มีค่ามากกว่า 7 จึงท่องลงมาทางขวา เทียบ 9 กับ 8 เนื่องจาก 9 มีค่ามากกว่า 8 จึงแทรกด้านขวาของ 8	
7. อ่านค่า 3 มาสร้างเป็นโหนด ③ แล้วนำไปแทรกในต้นไม้ โดยเทียบ 3 กับ 7 เนื่องจาก 3 มีค่าน้อยกว่า 7 จึงท่องลงมาทางซ้าย เทียบ 3 กับ 4 เนื่องจาก 3 มีค่าน้อยกว่า 4 จึงแทรกด้านซ้ายของ 4	

การดำเนินการ	ภาพของต้นไม้
8. อ่านค่า 2 มาสร้างเป็นโหนด (2) แล้วนำไปแทรกในต้นไม้ โดยเทียบ 2 กับ 7 เนื่องจาก 2 มีค่าน้อยกว่า 7 จึงท่องลงมาทางซ้าย เทียบ 2 กับ 4 เนื่องจาก 2 มีค่าน้อยกว่า 4 จึงท่องลงมาทางซ้าย เทียบ 2 กับ 3 เนื่องจาก 2 มีค่าน้อยกว่า 3 จึงแทรกด้านซ้ายของ 3	

จากตัวอย่างที่ 6.1 เมื่อดำเนินการแทรกข้อมูลจนกระทั่งข้อมูลหมดจะได้ไบนารีเสิร์ชทรีดังรูปที่ 6.21



รูปที่ 6.21 ภาพของไบนารีเสิร์ชทรีจากตัวอย่างที่ 6.1

จากลำดับขั้นตอนในการสร้างไบนารีเสิร์ชทรีเราสามารถนำมาเขียนเป็นฟังก์ชั่นในการนำโหนดใหม่ไปแทรกในไบนารีเสิร์ชทรีที่มี root ชี้อยู่ที่หัวได้ดังโปรแกรมที่ 6.4

โปรแกรมที่ 6.4 การเขียนฟังก์ชั่นเพื่อนำโหนดใหม่ที่ชี้โดย p ไปแทรกในไบนารีเสิร์ชทรีที่มี root ชี้อยู่ที่หัว

```
void  insertbst(int data)
 {
    node *p, *start, *prv;
    p = new node;
    p->data = data;
    p->LLink = NULL;
    p->RLink = NULL;
    if (root == NULL)
            root = p;
    else
    {
            start = root;
            while(start !=NULL)
            {
                 prv = start;
                if (start->data > p->data)
                     start= start->LLink;
                else
                    start = start->RLink;
            }
            if (prv->data > p->data)
                prv->LLink = p;
            else
                prv->RLink = p;
       }
 }
```

และเพื่อให้เข้าใจการทำงานของไบนารีเสิร์ชทรียิ่งขึ้น ขอให้พิจารณาโปรแกรมที่ 6.5 ซึ่งเป็นการใช้ฟังก์ชั่น insertbst เพื่อแทรกโหนดใหม่ในไบนารีเสิร์ชทรีและการใช้ฟังก์ชั่นสำหรับท่องเข้าไปในต้นไม้ในรูปแบบต่างๆ

โปรแกรมที่ 6.5 การใช้ฟังก์ชั่น insertbst เพื่อแทรกโหนดใหม่ในไบนารีเสิร์ททรีและการใช้ฟังก์ชั่นท่องเข้าไปในต้นไม้ในรูปแบบต่างๆ

```
#include <stdio.h>
#include <conio.h>

struct  node
{
     node  *LLink;
     int  data;
     node *RLink;
};
 node  *root;

void  insertbst(int data)
 {
    Node  *p, *start, *prv;

    p = new node;
    p->data = data;
    p->LLink = NULL;
    p->RLink = NULL;
    if (root == NULL)
         root = p;
    else
    {
         start = root;
         while(start !=NULL)
         {
              prv = start;
              if (start->data > p->data)
                start= start->LLink;
              else
                start = start->RLink;
            }
            if (prv->data > p->data)
               prv->LLink = p;
            else
               prv->RLink = p;
       }
 }
void  preorder(node *p)
{
   if (p != NULL)
   {
            printf("%d   ",p->data);
            preorder(p->LLink);
            preorder(p->RLink);
   }
}
```

โปรแกรมที่ 6.5 (ต่อ)

```
void  inorder(node *p)
{
   if (p != NULL)
   {
     inorder(p->LLink);
     printf("%d   ",p->data);
     inorder(p->RLink);
   }
}

void  postorder(node *p)
{
   if (p != NULL)
   {
     postorder(p->LLink);
     postorder(p->RLink);
     printf("%d   ",p->data);
   }
}
```

```
main()
{
   clrscr();
   root =NULL;
   insertbst(6);
   insertbst(9);
   insertbst(4);
   insertbst(5);
   insertbst(10);
   insertbst(7);
   insertbst(3);
   insertbst(8);
   printf("\nPreorder Traversal  =  ");
   preorder(root);
   printf("\n\nInorder Traversal   =  ");
   inorder(root);
   printf("\n\nPostorder Traversal =  ");
   postorder(root);
   getch();
   return 0;
}
```

ผลรัน

Preorder Traversal = 6 4 3 5 9 7 8 10

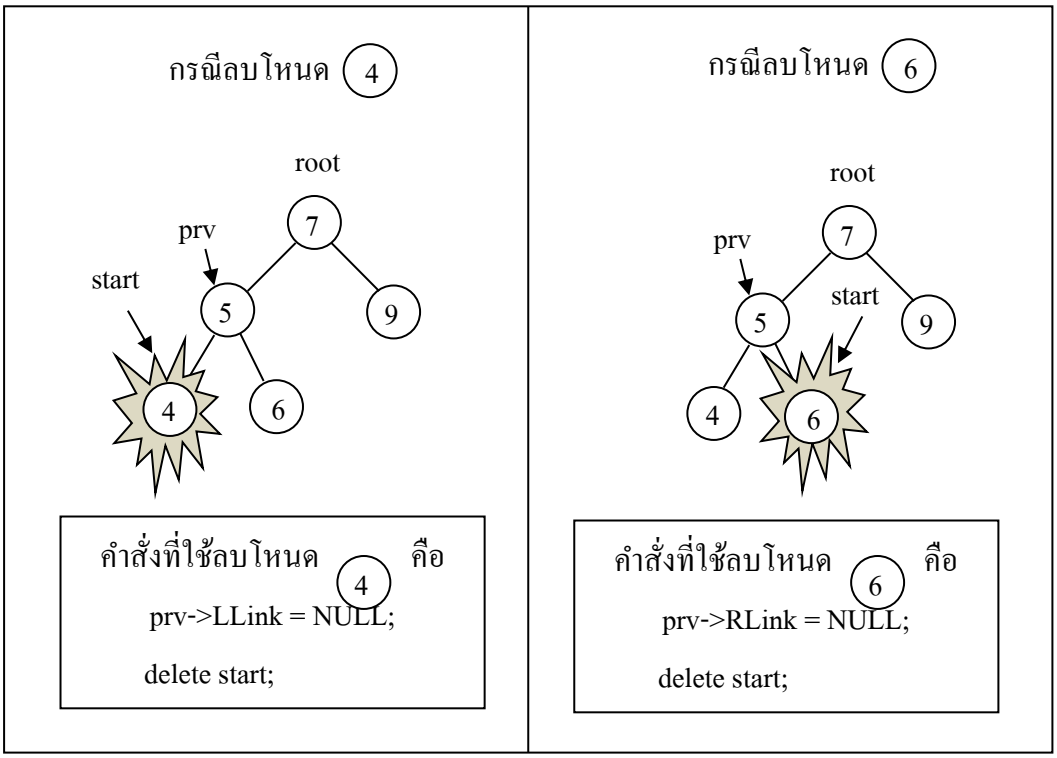
Inorder Traversal = 3 4 5 6 7 8 9 10

Postorder Traversal = 3 5 4 8 7 10 9 6

การลบโหนดออกจากไบนารีเสิร์ชทรี

การลบโหนดออกจากไบนารีเสิร์ชทรีจำเป็นต้องค้นหาตำแหน่งโหนดที่ต้องการลบเป็นประการแรก ซึ่งการลบโหนดออกจากไบนารีเสิร์ชทรีมี 3 กรณี คือ

1. กรณีการลบโหนดที่ปลาย จะใช้พอยน์เตอร์ 2 ตัว คือ ใช้ start เป็นพอยน์เตอร์เพื่อท่องเข้าไปหาโหนดที่ต้องการลบ และให้ prv ตามติดหลัง start เมื่อพบแล้วนั่นคือ start จะชี้ไปยังโหนดที่ต้องการลบ ส่วน prv จะชี้ตามติดหลัง start จากนั้นก็ทำการเปลี่ยนแปลงพอยน์เตอร์ที่ชี้โหนดและทำการลบโหนดที่ start ชี้อยู่ออกไป เพื่อความเข้าใจมากยิ่งขึ้นให้พิจารณาภาพจำลองการลบโหนดที่ปลาย ดังรูปที่ 6.22



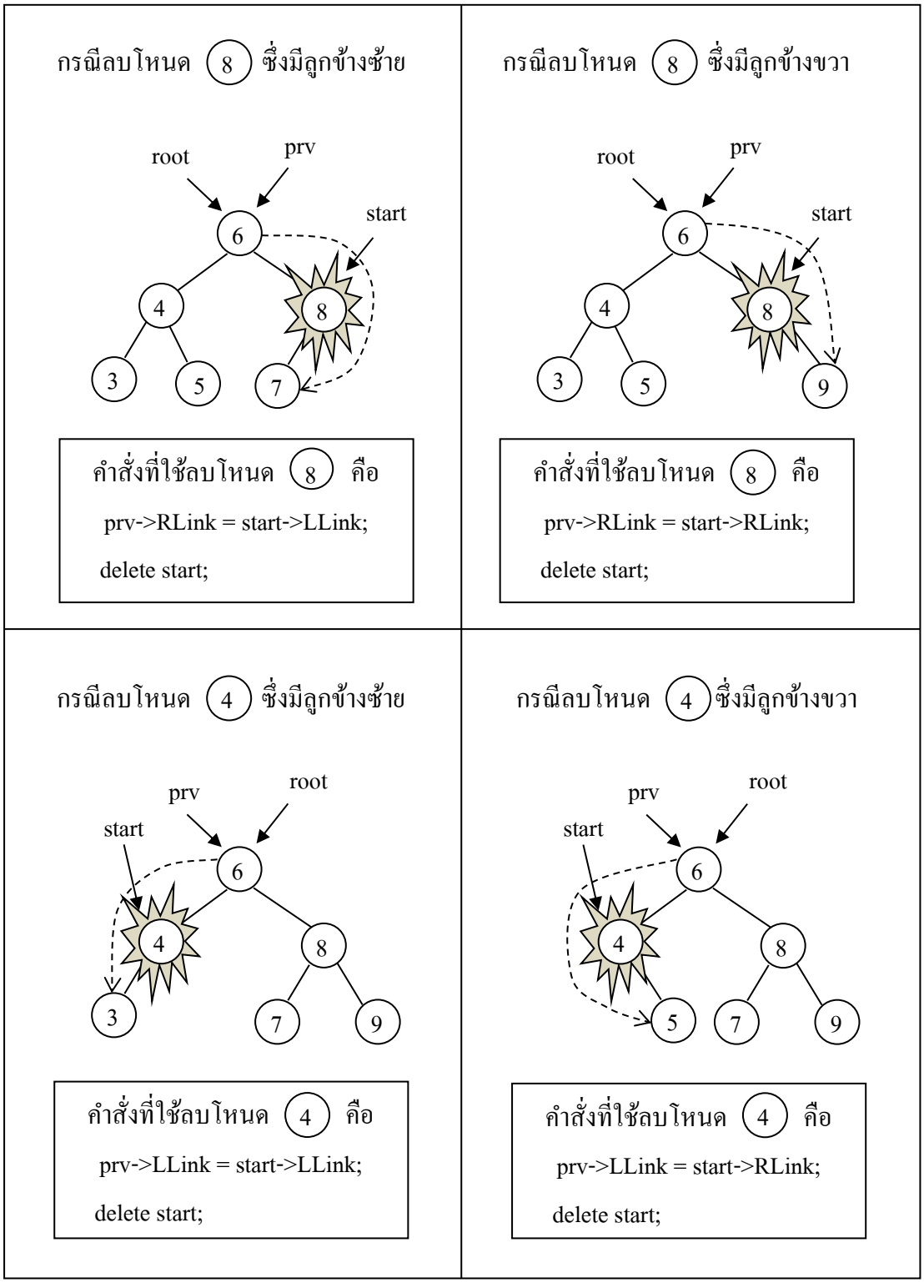
รูปที่ 6.22 ภาพจำลองการลบโหนดที่ปลาย

และหากต้นไม้มีเพียงโหนดเดียว คือ โหนดที่เป็นรูท หากต้องการลบโหนดนี้ จะต้องเปลี่ยนค่าของ root ด้วย โดยใช้คำสั่ง

```
root = NULL;
```

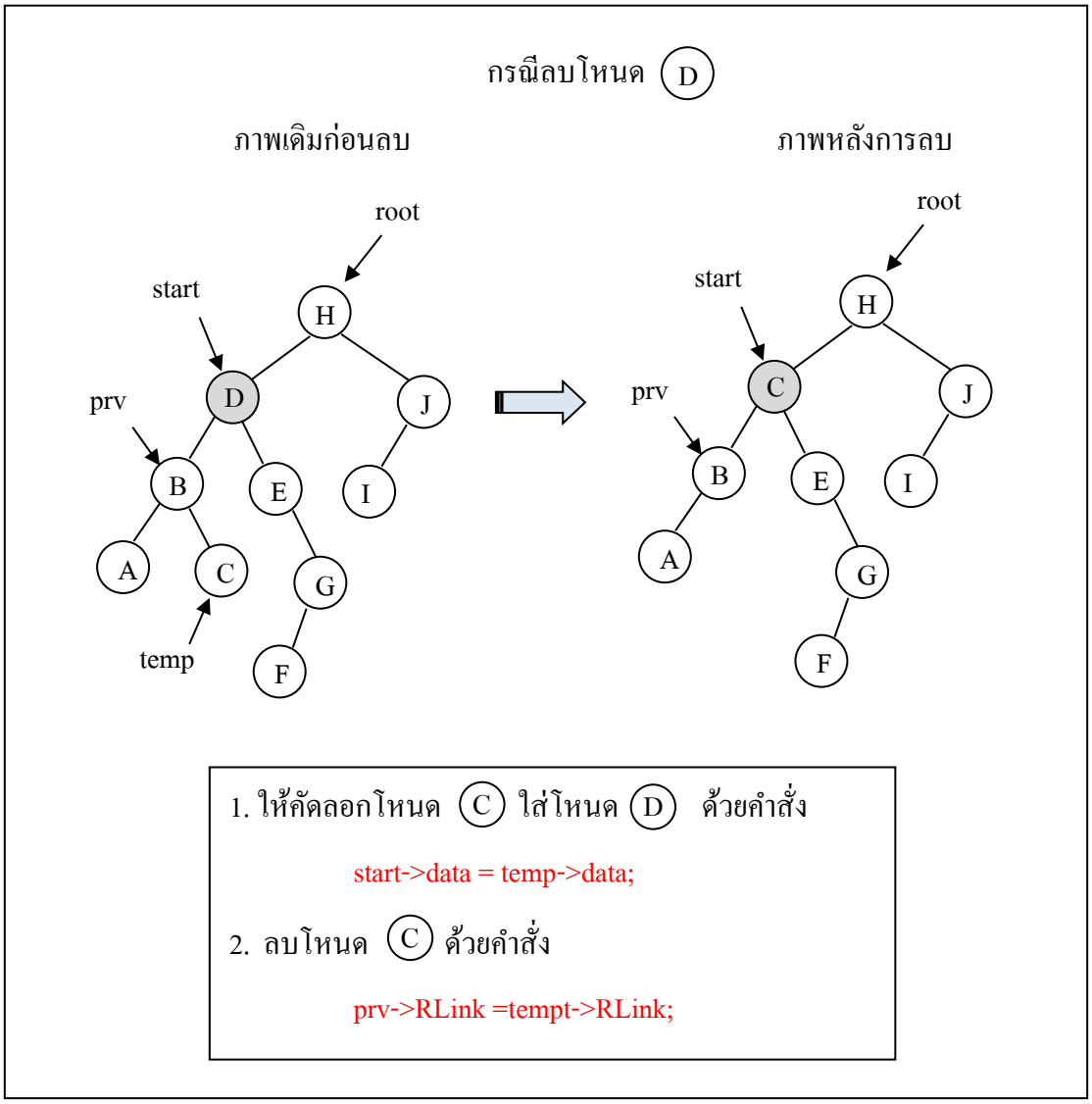
นั่นหมายความว่าเมื่อต้นไม้มีเพียงโหนดเดียว ดังนั้นเมื่อลบโหนดนี้แล้วต้นไม้ย่อมมีสภาพเป็นต้นไม้ว่างเปล่า (Empty Tree)

2. กรณีการลบโหนดที่มีลูกเพียงหนึ่งโหนด ซึ่งอาจเป็นลูกด้านซ้ายหรือด้านขวาก็ได้ ให้พิจารณาภาพจำลองการลบ ดังรูปที่ 6.2



รูปที่ 6.23 ภาพจำลองการลบโหนดที่มีลูกเพียงหนึ่ง

3. กรณีการลบโหนดที่มีลูกสอง มีขั้นตอนการลบดังนี้คือ ให้ทำการท่องไปหาโหนดที่อยู่ก่อนหน้าโหนดที่ต้องการลบ จากนั้นให้ทำการคัดลอกข้อมูลจากโหนดก่อนหน้าที่ต้องการลบทั้งหมดให้กับโหนดที่จะลบ แล้วจัดการลบโหนดก่อนหน้าโหนดที่ต้องการลบนั้นทิ้งไปแทน เพื่อให้เข้าใจมากยิ่งขึ้นให้พิจารณา ดังรูปที่ 6.24 ต้องการลบโหนดที่ start ชี้อยู่

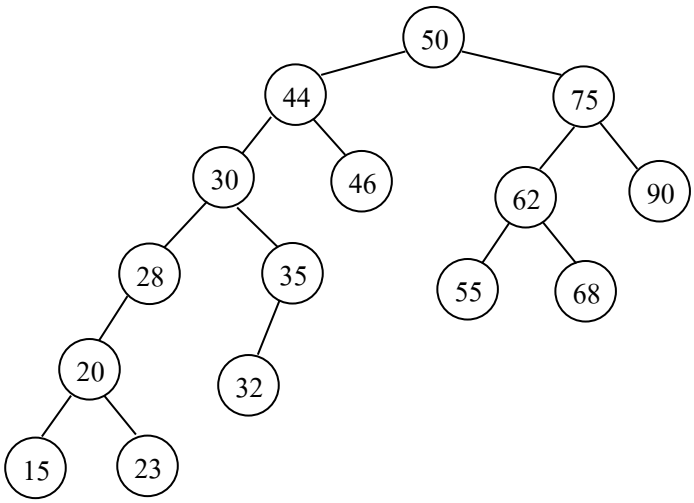


รูปที่ 6.24 ภาพจำลองการลบโหนดที่มีลูกสอง

แบบฝึกหัดหน่วยที่ 6

ตอนที่ 1 จงตอบคำถามต่อไปนี้

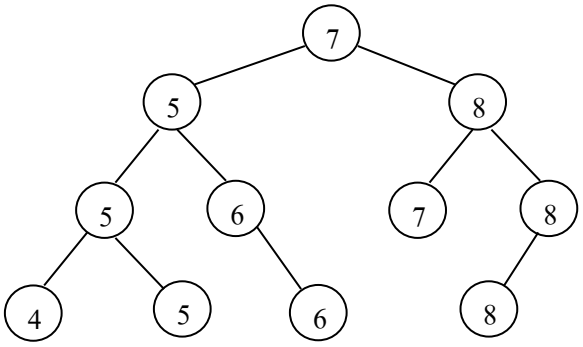
1. จงอธิบายลักษณะโครงสร้างข้อมูลแบบต้นไม้
2. จงอธิบายลักษณะของไบนารีทรี
3. จงอธิบายลักษณะโครงสร้างข้อมูลแบบไบนารีทรี
4. ไบนารีเสิร์ชทรีมีคุณสมบัติอย่างไร
5. จากไบนารีทรีที่กำหนดให้ จงหาผลลัพธ์จากการท่องเข้าไปในไบนารีทรี แบบพรีออเดอร์ แบบอินออเดอร์ และแบบโพสต์ออเดอร์



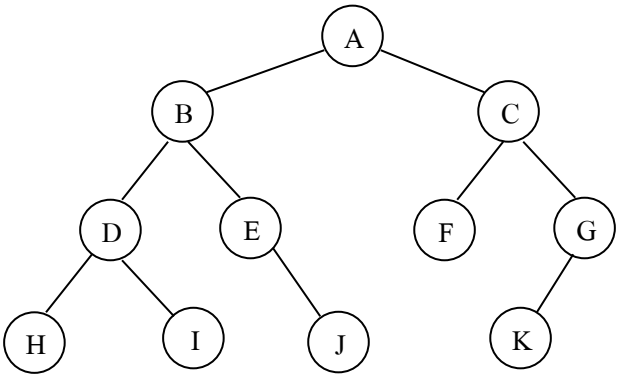
6. จงสร้างไบนารีเสิร์ชทรี จากข้อมูลที่กำหนดให้ต่อไปนี้

50 60 80 46 57 30 75 30 75 38 25 84

7. จากไบนารีเสิร์ชทรีที่กำหนด จงวาดรูปไบนารีเสิร์ชทรีใหม่ หลังจากได้ดำเนินการลบโหนด 40, โหนด 64 และโหนด 80 ออกจากทรี



ตอนที่ 2 จงเติมคำลงในช่องว่างให้ถูกต้อง



จากไบนารีทรีที่กำหนดให้

1. รูทโหนด คือ.....
2. โหนดพ่อแม่ คือ.....
3 โหนดลูกของ C คือ.....
4 โหนดพี่น้อง คือ.....
5 โหนดใบ คือ.....
6 ต้นไม้ย่อย คือ.....
7. โหนดลูกหลานของ B คือ.....
8. บรรพบุรุษของ K คือ.....
9. ดีกรีของ B เท่ากับ.....
10. ดีกรีทั้งหมดของต้นไม้ เท่ากับ.....

ตอนที่ 3 จงอธิบายความหมายของศัพท์ต่อไปนี้

1. Root Node
2. Leaf Node
3. Degree
4. Branch
5. Sibling Node
6. Level
7. child Node
8. Parent Node
9. Ancestor
10. Subtree
11. Descendant

กิจกรรมฝึกปฏิบัติหน่วยที่ 6

จุดประสงค์

1. เพื่อให้นักศึกษามีความรู้ความเข้าใจเกี่ยวกับโครงสร้างข้อมูลแบบต้นไม้
2. เพื่อให้นักศึกษาเข้าใจการทำงานของโครงสร้างข้อมูลแบบต้นไม้
3. เพื่อฝึกทักษะการเขียนโปรแกรมคอมพิวเตอร์จัดการข้อมูลในโครงสร้างข้อมูลแบบต้นไม้

กิจกรรม

1. ให้นักศึกษาจับคู่กัน
2. ให้นักศึกษาแต่ละคู่ร่วมกันศึกษาตัวอย่างโปรแกรมที่ 6.5 โดยทำการป้อนโปรแกรมที่กำหนดโดยใช้ Editor ของภาษาซี ทำการทดสอบโปรแกรมโดยทำการคอมไพล์และ RUN โปรแกรมเพื่อดูผลลัพธ์ที่ได้ และอธิบายคำสั่งแต่ละบรรทัดในโปรแกรมว่าเป็นคำสั่งให้ทำอะไร และวาดภาพจำลองโครงสร้างข้อมูลแบบต้นไม้ที่ได้
3. ให้นักศึกษาน้อมนำหลักปรัชญาเศรษฐกิจพอเพียงมาบูรณาการในการทำกิจกรรม โดยให้ทำกิจกรรมด้วยความรอบคอบ ใช้หลักเหตุผลในการคิดวิเคราะห์งาน และดำเนินงานตามขั้นตอนด้วยความระมัดระวังเพื่อความปลอดภัย