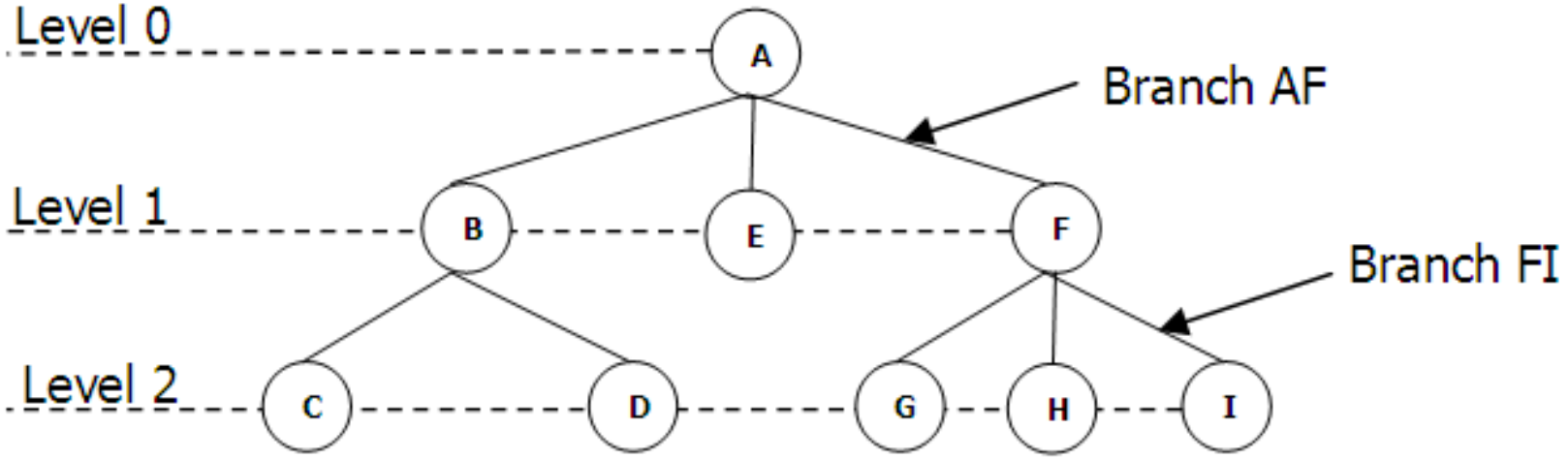


ค่ายโอลิมปิกวิชาการ
ศูนย์มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

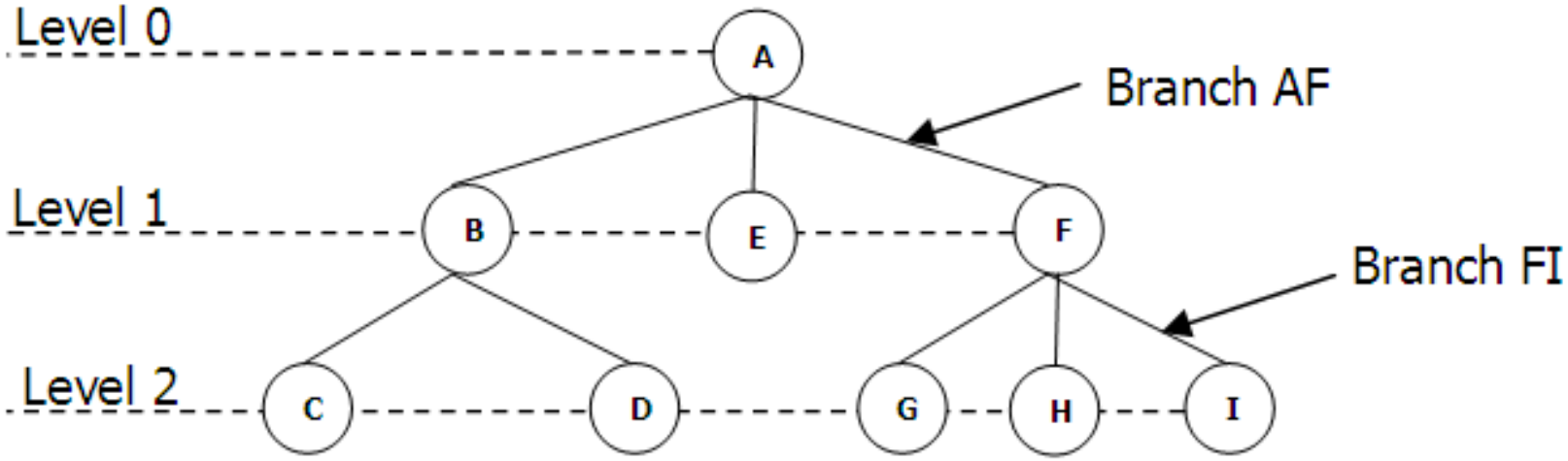
Binary trees and Traversals

ต้นไม้ (Tree)

- Tree เป็นโครงสร้างชนิดไม่เชิงเส้น (Non-linear) มีลักษณะเป็น recursive ประกอบไปด้วยสมาชิกที่เรียกว่า Node และมีเส้นที่เชื่อมระหว่าง Node ที่เรียกว่า branch คำสำคัญที่เกี่ยวกับ Tree มีดังนี้

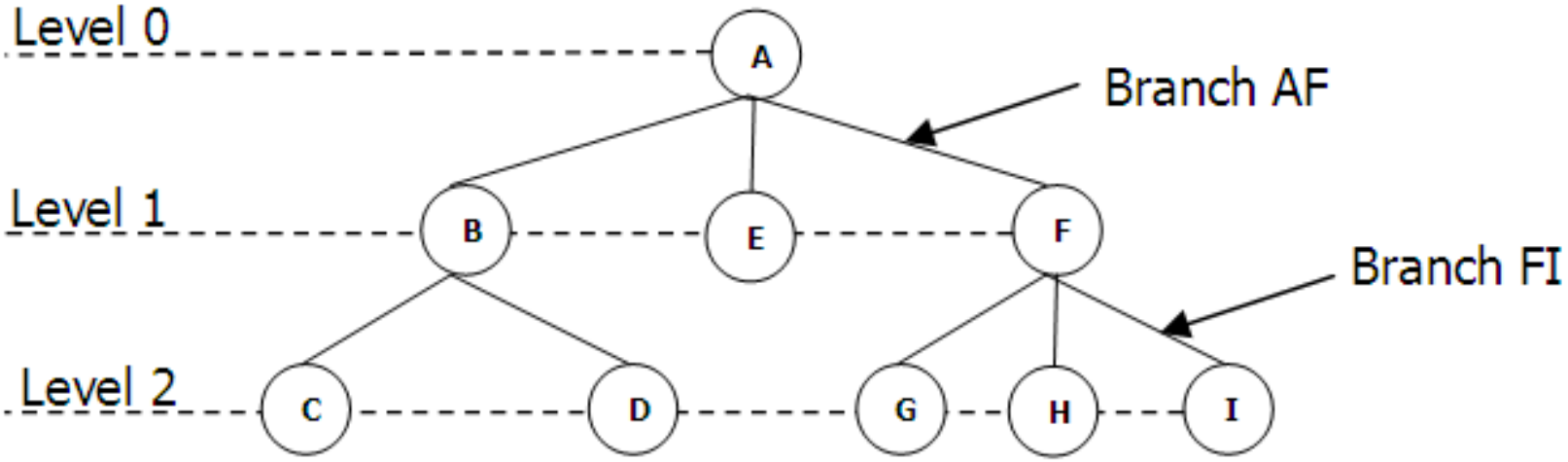


คุณลักษณะของต้นไม้



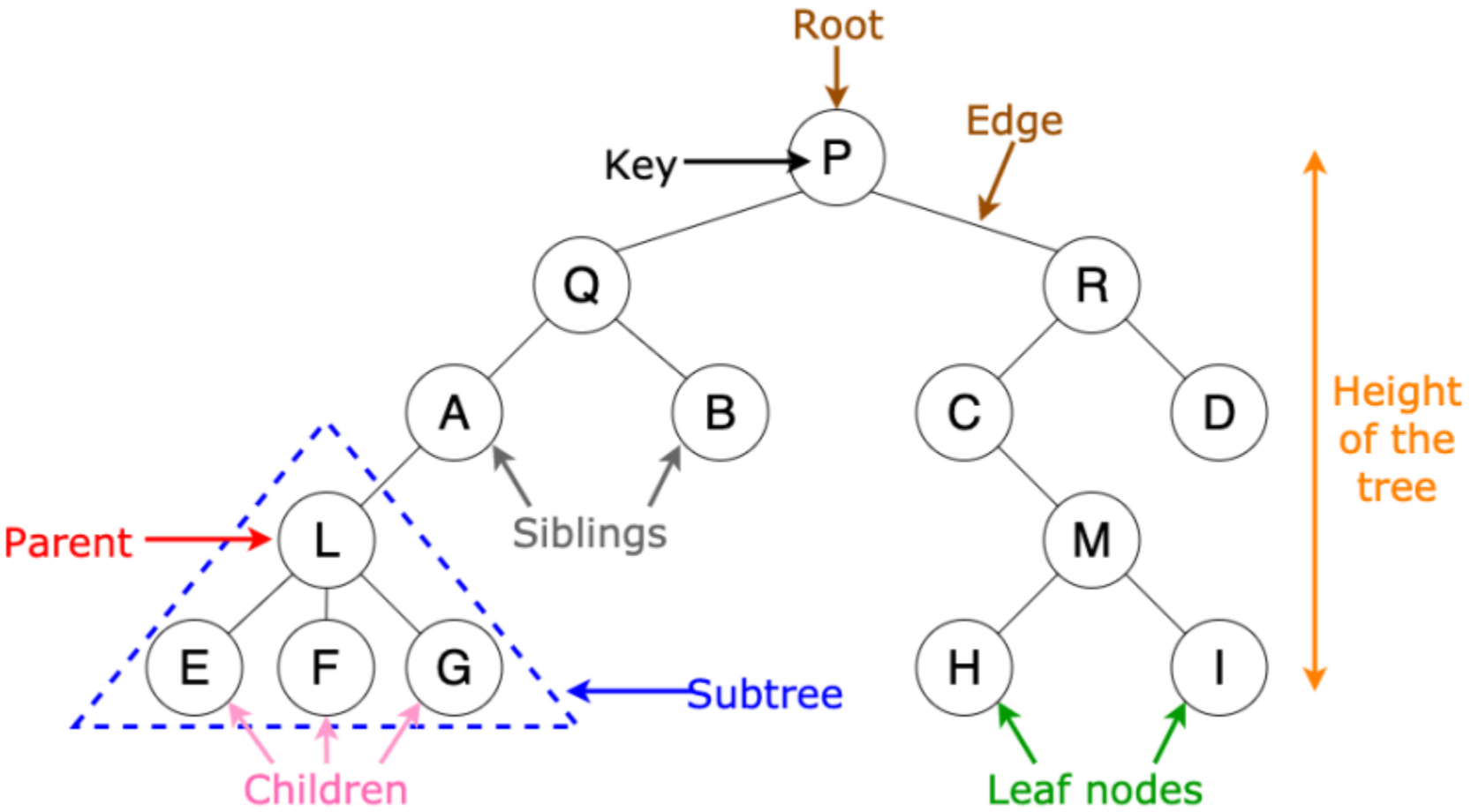
- Root Node คือ โหนดที่อยู่บนสุดของต้นไม้
- Leaf Node คือ โหนดที่ไม่มีลูกหรือโหนดอื่นต่อ เรียกอีกอย่างหนึ่งว่า External Node
- Internal Node คือ โหนดที่ไม่ใช่ Root และ Leaf Node
- Depth คือ ความยาวจาก Root node ถึง Node ที่สนใจ
- Height คือ ความยาวจาก Node ที่สนใจถึง Leaf Node ที่ลึกที่สุดที่มี Node ที่สนใจเป็น Parent

ตัวอย่าง



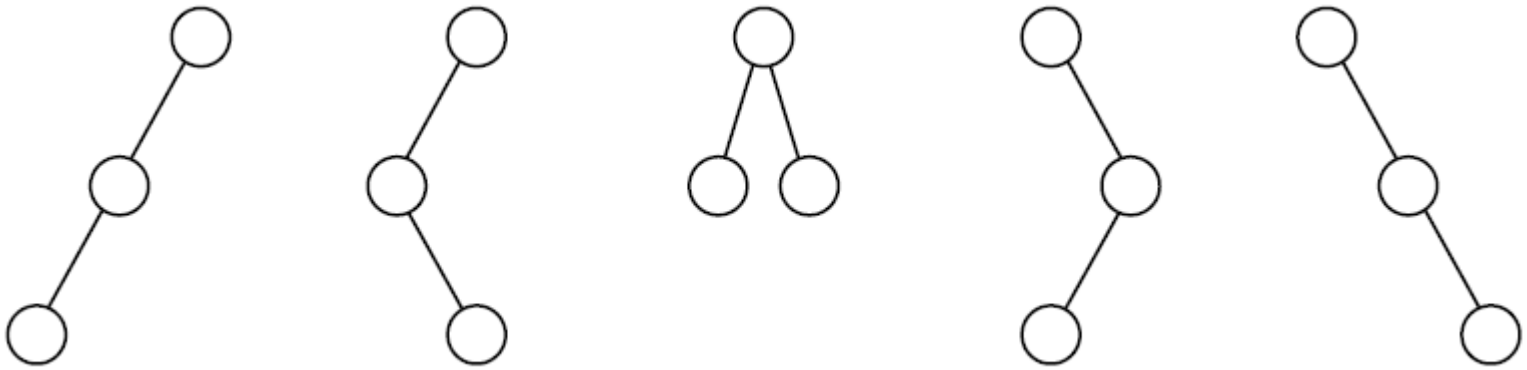
Root Node: A	**Sibling Node:** {B, E, F}, {C, D}, {G, H, I}
Parents Node: A, B, F	**Leaves Node:** C, D, E, G, H, I
Child Node: B, E, F, C, D, G, H, I	**Internal Node:** B, F

Subtree in Tree



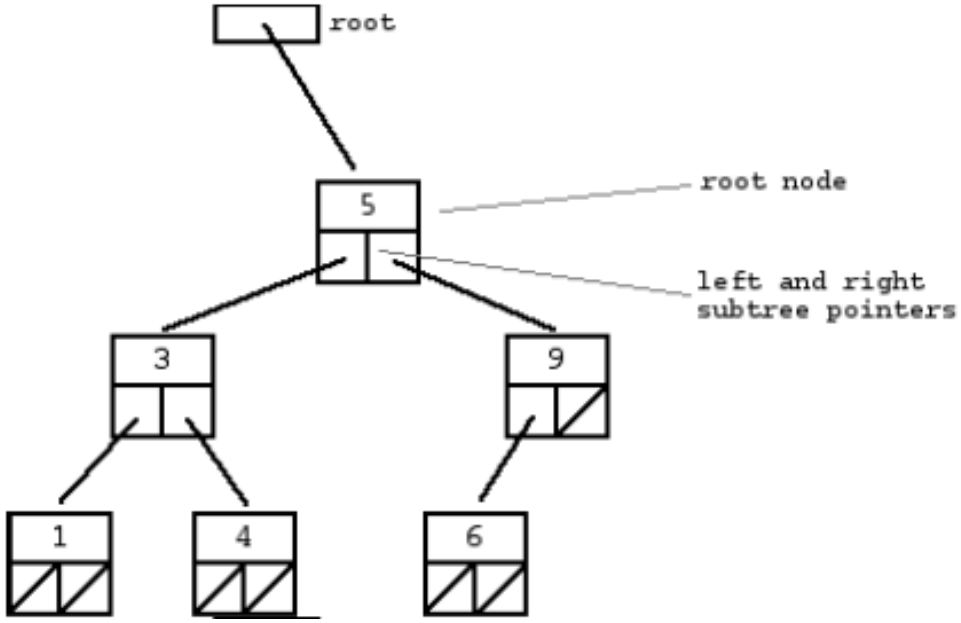
ต้นไม้ทวิภาค (Binary tree)

- คือโครงสร้างต้นไม้ที่มีแต่ละโหนดจะมีโหนดลูก (child node) ได้สูงสุดไม่เกิน 2 ตัว
  - โหนดลูกด้านซ้าย (left child node)
  - โหนดลูกด้านขวา (right child node)



โครงสร้างต้นไม้ทวิภาค (Binary tree)

```
struct node {
    int data;
    struct node* left;
    struct node* right;
}
```

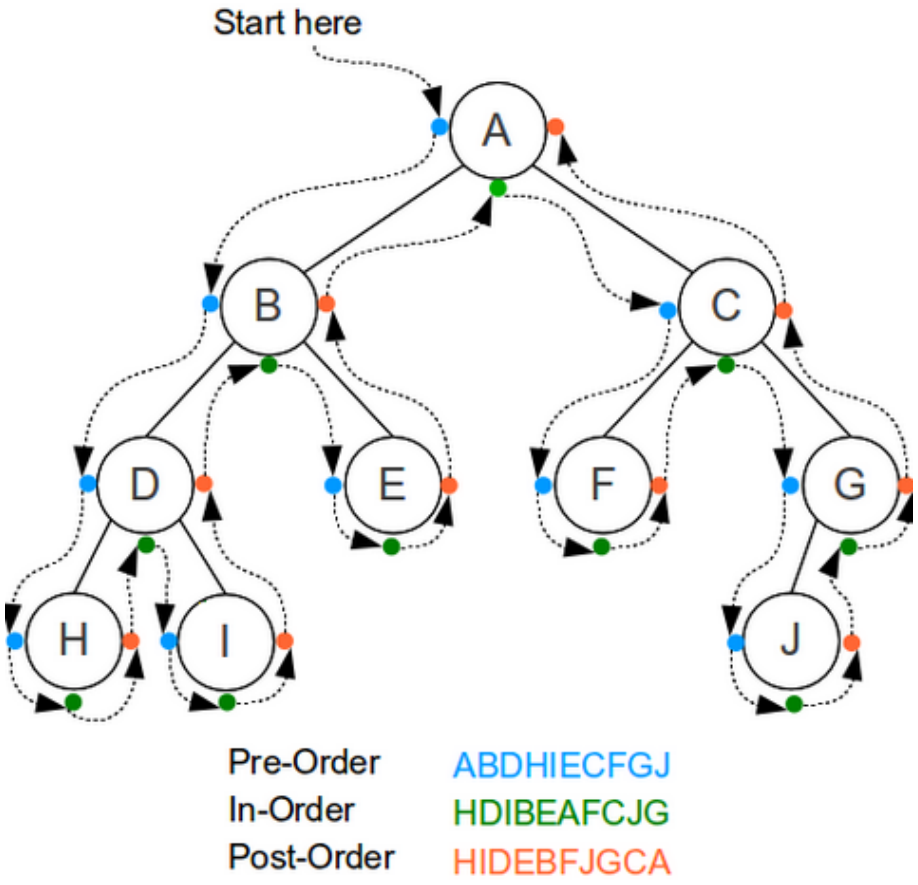



```
struct node* NewNode(int data) {
  struct node* node = new(struct node);
  node->data = data;
  node->left = NULL;
  node->right = NULL;

  return(node);
}
```

การท่องต้นไม้ทวิภาค

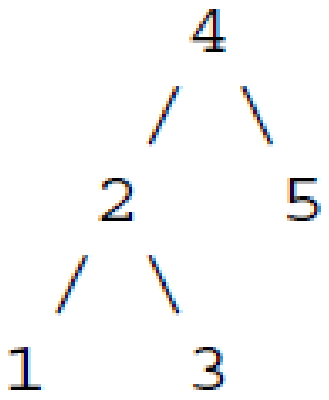
- การท่องแต่ละแบบจะเหมาะกับการหาคำตอบของปัญหาที่แตกต่างกัน
- หลักการท่องในต้นไม้ จะเริ่มจากโหนดราก (Root Node) ท่องไปสู่โหนดอื่นๆ ตามความสัมพันธ์แบบ Parent-Child เพื่อทำการประมวลผลข้อมูลที่อยู่ในโหนดนั้นๆ จนครบทุกโหนด
- สำหรับต้นไม้ทวิภาค เราสามารถเข้าถึงแต่ละโหนดในต้นไม้ ได้ 3 รูปแบบ คือ
- Preorder
  - รูตโหนด (root) -> ต้นไม้ย่อยด้านซ้าย (left sub tree) -> ต้นไม้ย่อยด้านขวา (right sub tree)
- Inorder
  - ต้นไม้ย่อยด้านซ้าย (left sub tree) -> รูตโหนด (root) -> ต้นไม้ย่อยด้านขวา (right sub tree)
- Postorder
  - ต้นไม้ย่อยด้านซ้าย (left sub tree) -> ต้นไม้ย่อยด้านขวา (right sub tree) -> รูตโหนด (root)

รูปแบบการท่องต้นไม้ไบนารี



Pre-Order: Root-Left-Right
In-Order: Left-Root-Right
Post-Order: Left-Right-Root

Preorder traversal

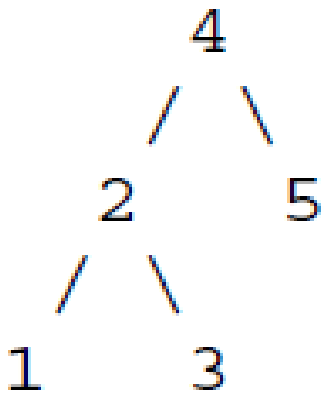


ผลลัพธ์ของ preorder

“4 2 1 3 5”

```
void preorder(struct node *p) {
  if(p==NULL) return;
  printf(“%d”, p->data);        // print root
  preorder(node->left);         // visit left subtree
  preorder(node->right);        // visit right subtree
```

Inorder traversal

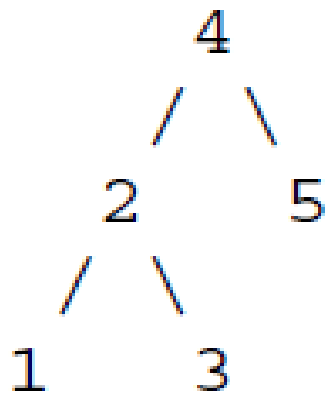


ผลลัพธ์ของ inorder

“1 2 3 4 5”

```
void inorder(struct node *p) {
  if(p==NULL) return;
  inorder(node->left);        // visit left subtree
  printf(“%d”, p->data);      // print root
  inorder(node->right);       // visit right subtree
```

Postorder traversal

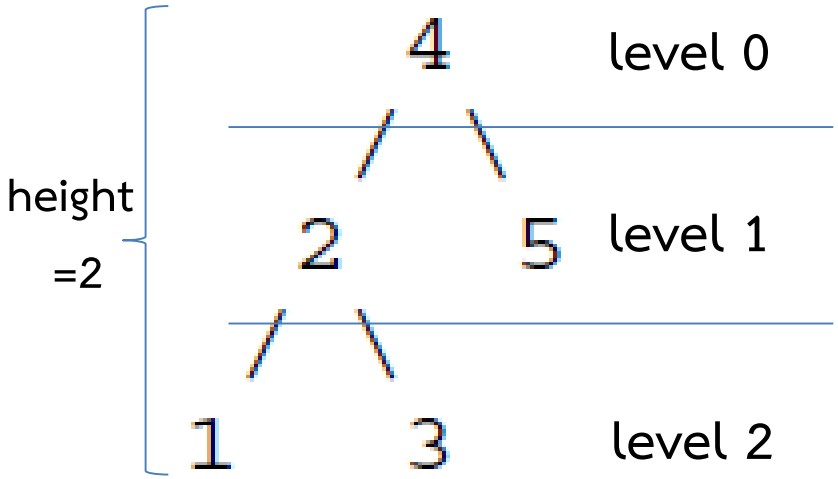


ผลลัพธ์ของ postorder

“1 3 2 5 4”

```
void postorder(struct node *p) {
  if(p==NULL) return;
  postorder(node->left);       // visit left subtree
  postorder(node->right);      // visit right subtree
  printf(“%d”, p->data);       // print root
```

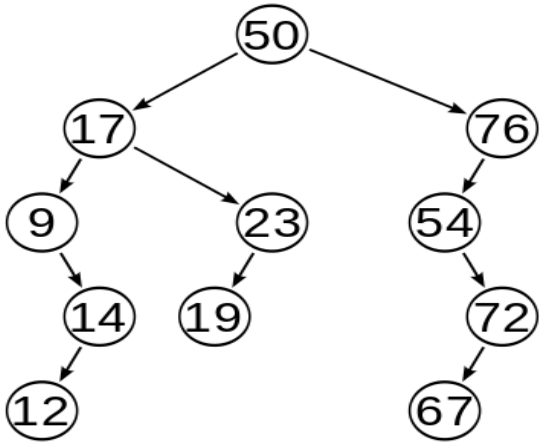
การคำนวณความสูงของต้นไม้

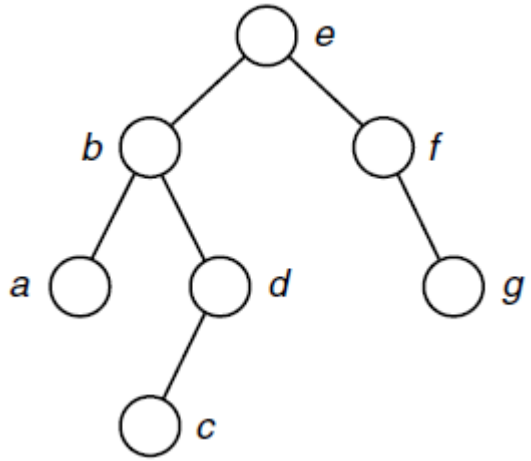


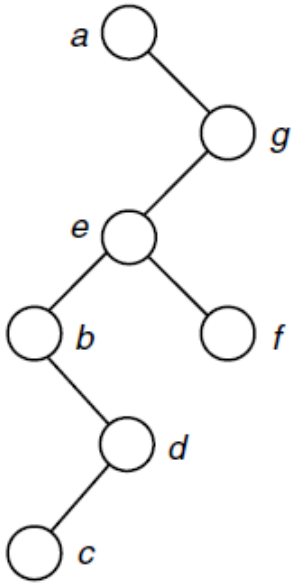

```
int height(struct node *r)
{ if(r == null)
        return 0;

 else {
  int ldepth=height(r->left);
  int rdepth=height(r->right);

  if(ldepth>rdepth) ledepth+1;
  else rdepth+1;
 }
}
```

ต้นไม้ค้นหาแบบทวิภาค (Binary Search Tree)

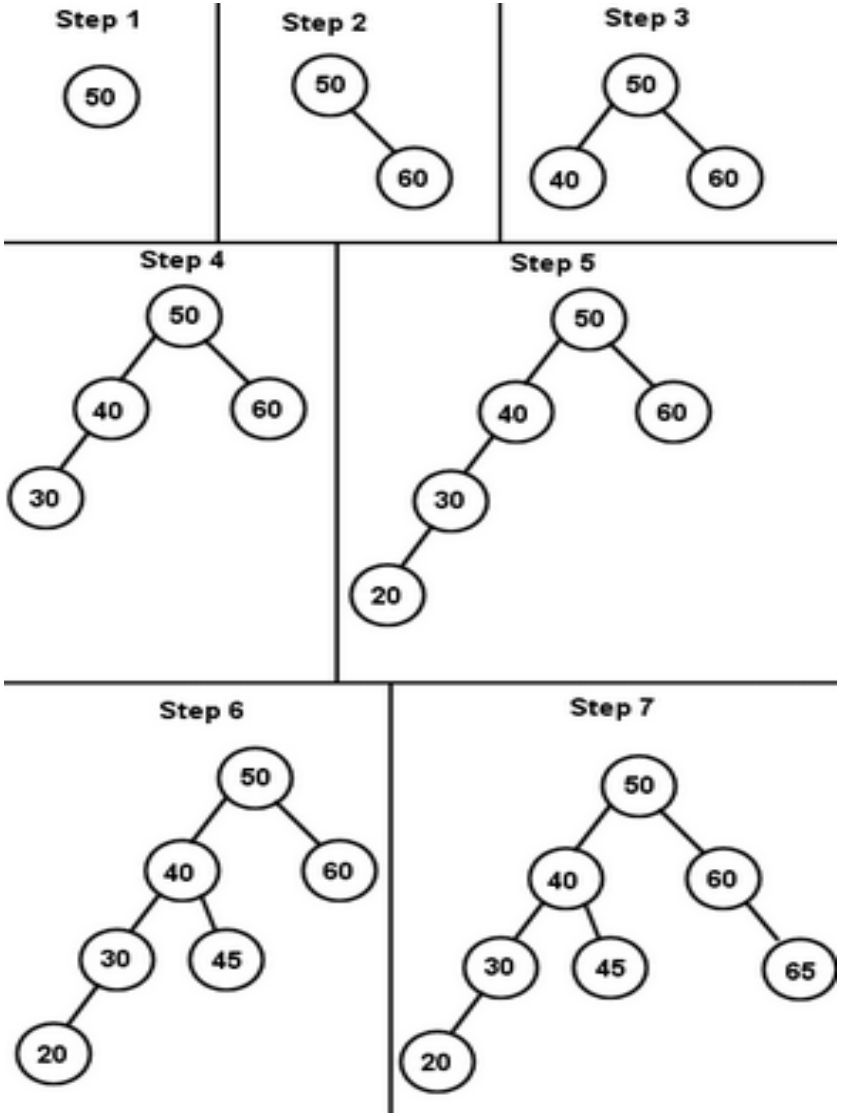
- ต้นไม้ค้นหาไบนารี คือต้นไม้ที่มีจำนวนลูก (children) สูงสุดไม่เกิน 2 และมีคุณสมบัติดังนี้ สำหรับแต่ละโหนด m ในต้นไม้ไบนารี T
  - สำหรับ X ซึ่งเป็นโหนดใดๆ ในต้นไม้ย่อยด้านซ้าย TL(m) : X < m
  - สำหรับ Y ซึ่งเป็นโหนดใดๆ ในต้นไม้ย่อยด้านซ้าย TL(m) : Y > m







การเพิ่มโหนดใน BST



สมมติว่าข้อมูลอินพุตคือ

50 60 40 30 20 45 65

```
void insert(struct node *p, int key)
{ if(key < p->data)
    {  if(p->left== NULL)
          p->left = new_node(key);
       else insert(p->left, key);
    }
 else
     {  if(p->right== NULL)
          p->right = new_node(key);
       else insert(p->right, key);
    }
}
```

การค้นหาข้อมูลในต้นไม้ค้นหาทวิภาค

```
int seek(struct node *p, int target)
{ if(node == NULL)
     return -1;
  else
    {  if(target == p->data)
          return 1;
       else
         {  if(target < p->data)
                return seek(p->left, target);
            else
                return seek(p->right, target);
         }
}
```

ค้นหาค่ามากที่สุดและน้อยที่สุด

```
int find_min(struct node *p)
{   if(p==NULL)
        return p->data;
    return find_min(p->left);
}
```

```
int find_max(struct node *p)
{   if(p==NULL)
        return p->data;
    return find_max(p->right);
}
```

การลบโหนดในต้นไม้ค้นหาไบนารี

- แบ่งออกเป็น 3 กรณี

1. โหนดที่ต้องการลบเป็นโหนดใบ (leave node)

   - สามารถลบทิ้งได้ทันที

2. โหนดที่ต้องการลบมีโหนดลูก 1 โหนด

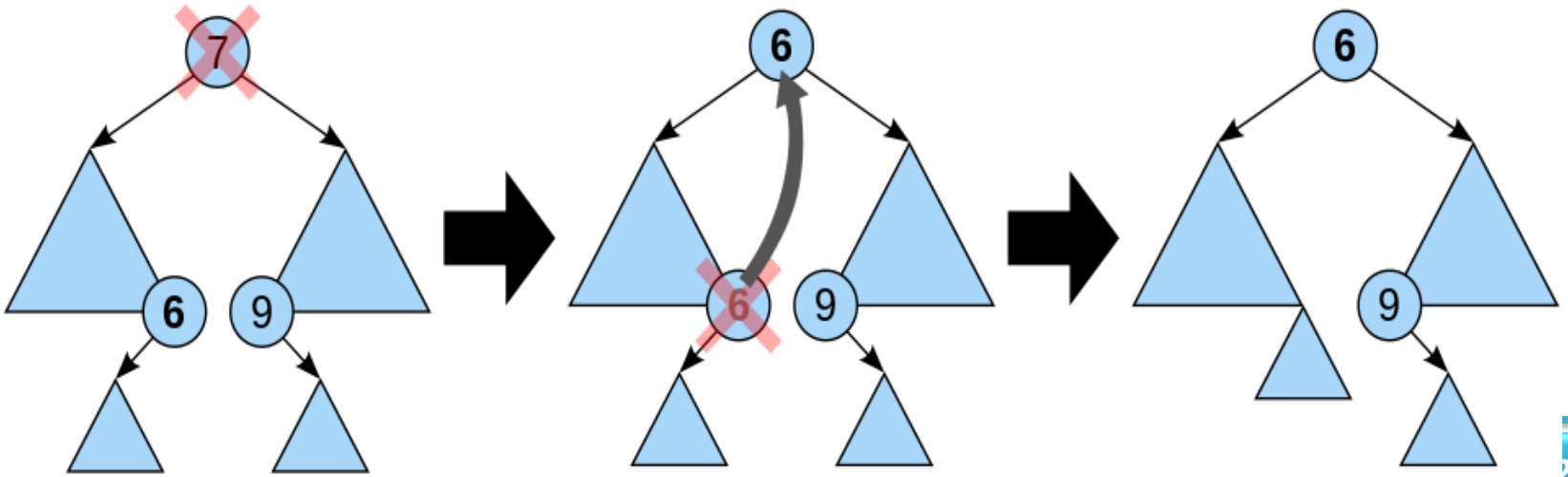
   - แทนที่โหนดที่ต้องการลบด้วยโหนดลูก

3. **โหนดที่ต้องการลบมีโหนดลูก 2 โหนด**

การลบโหนดซึ่งมีโหนดลูก 2 ตัว

- ทำได้ 2 วิธี

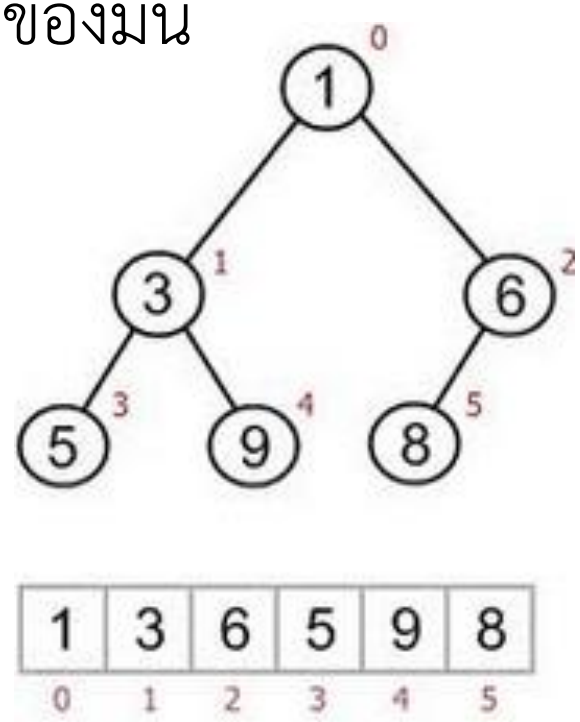
1. แทนที่โหนดที่ต้องการลบด้วย โหนดที่มีคีย์ที่มากที่สุดในต้นไม้ย่อยด้านซ้าย (Find Max)
2. แทนที่โหนดที่ต้องการลบด้วย โหนดที่มีค่าน้อยที่สุดในต้นไม้ย่อยด้านขวา (Find Min)



```
void delete(struct node *m, int key)
{  struct node *cur, *successor;
     cur = seek(m, key);
 if(cur->left !=NULL && cur->right !=NULL)
    { successor = find_min(cur->right);
      cur->data = successor->data;
      free(successor);
    }
  else if(cur->left != NULL)
          cur = cur->left;
  else if(cur->right != NULL)
          cur = cur->right;
   else  cur = NULL;
}
```

An array implementation

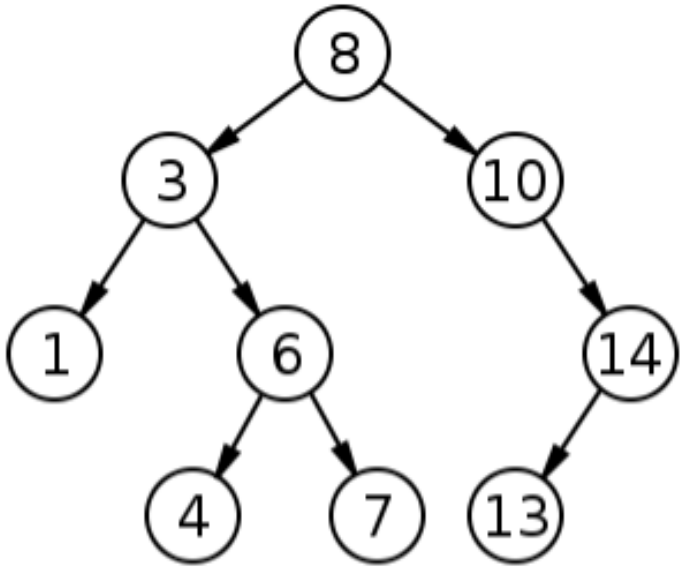
- เราสามารถใช้อาร์เรย์ในการจัดเก็บต้นไม้ค้นหาทวิภาคได้
- คำนวณตำแหน่งในอาร์เรย์สัมพันธ์กับโหนดพ่อแม่และโหนดลูกของมัน



Parent(i) = i/2

Left chid of i = 2i + 1

Right child of i = 2i + 2



0	1	2	3	4	5	6	7	8	9	10	11
8	3	10	1	6	14				4	7	13

```
find_BST(key, i, A[])
{    if (A[i] == key)
          printf("%d",A[i]);
     else if(key < A[i])
           find_BST(key, 2*i + 1, A);
     else if(key > A[i])
           find_BST(key, 2*i + 2, A);
     else
            printf("not found");
}
```