CS221 Data Structure

Chapter 7

Trees

By Dr. Paween Khoenkaw
Computer Science MJU

Trees:

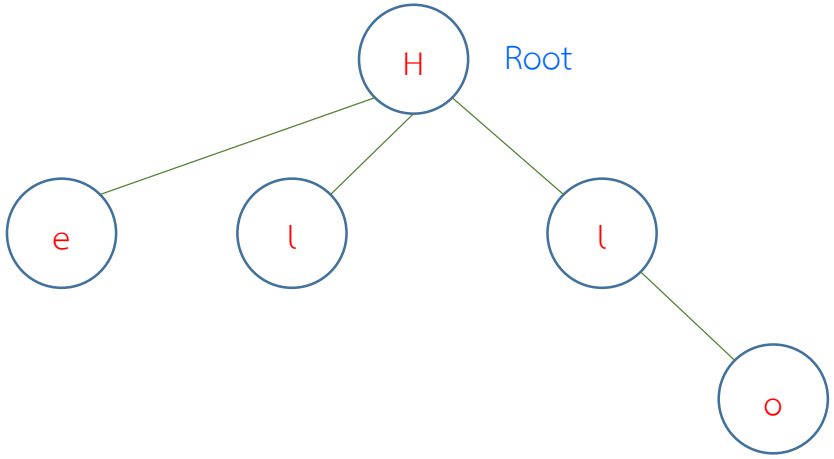
Tree หรือ ต้นไม้ เป็น แบบชนิดข้อมูลนามธรรม ประเภทหนึ่ง มีลักษณะการเรียงเป็นกิ่งก้านสาขาแตกแขนงออกไป จะไม่มีวงวน (loop) โยงในสมาชิกตัวต่าง ๆ โดยสมาชิกจะถูกเก็บไว้ในประเภทข้อมูลชนิดวัตถุ (Object) หรือโครงสร้าง (Structure) เรียกว่าปม (node) ซึ่งจะมีตัวแปรซึ่งเก็บตัวชี้ (Pointer) ไปยังปมอื่น ๆได้

ส่วนประกอบของต้นไม้จะประกอบด้วย

1) Node หรือปม คือสมาชิกของต้นไม้ใช้สำหรับเก็บข้อมูล แทนด้วยวงกลม

2) Edge หรือ กิ่ง ใช้ในการเชื่อม node บางครั้งเรียกว่า path แทนด้วยเส้น

3) ต้นไม้จะโตจาด้านบนลงไปด้านล่าง node ที่อยู่บนสุดจะเรียกว่า Root หรือราก



Trees:

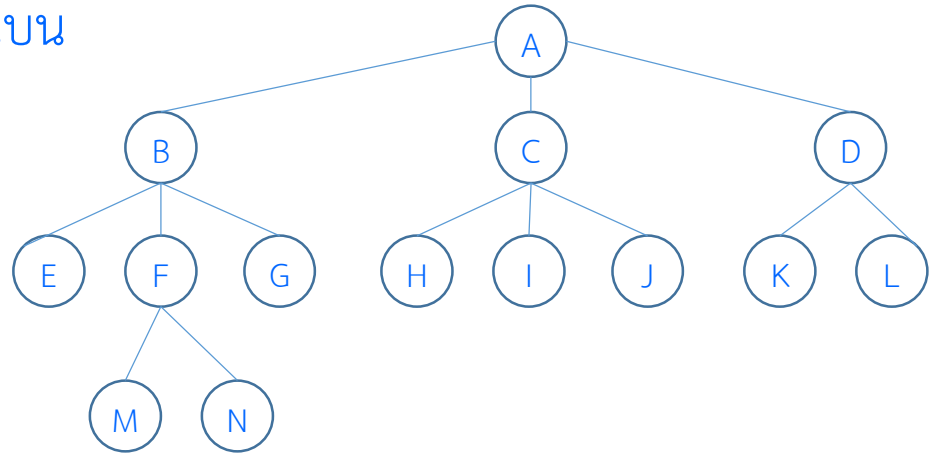
โครงสร้างข้อมูลแบบต้นไม้ นั้นมีต้นกำเนิดมาจากคณิตศาสตร์ในสาขาทฤษฎีกราฟ เพื่อให้เข้าใจตรงกัน จึงนามคุณลักษณะและคุณสมบัติของต้นไม้ดังต่อไปนี้

Parent node หรือปมพ่อ ใช้เรียก Node ที่อยู่ด้านบน

ตัวอย่างเช่น

B เป็น Parent ของ E,F,G

F เป็น Parent ของ M,N



Trees:

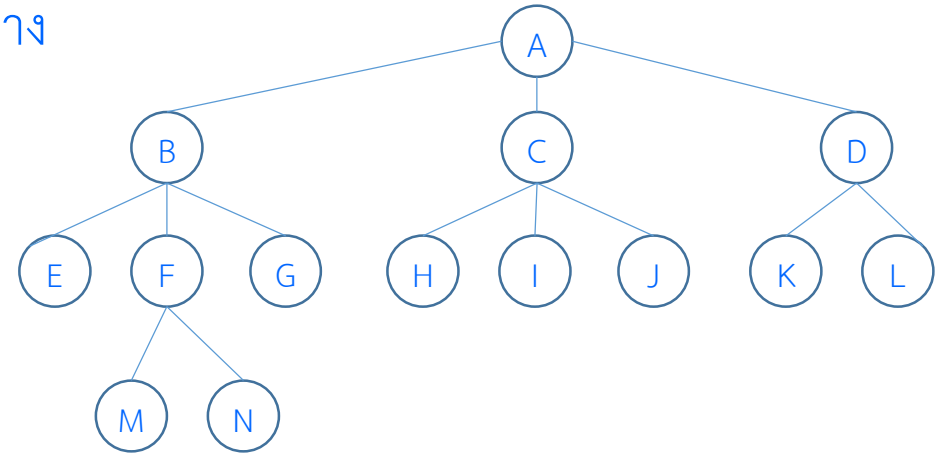
โครงสร้างข้อมูลแบบต้นไม้ นั้นมีต้นกำเนิดมาจากคณิตศาสตร์ในสาขาทฤษฎีกราฟ เพื่อให้เข้าใจตรงกัน จึงนามคุณลักษณะและคุณสมบัติของต้นไม้ดังต่อไปนี้

Child node หรือปมลูก ใช้เรียก Node ที่อยู่ด้านล่าง

ตัวอย่างเช่น

D เป็น Child node ของ A

K,L เป็น Child node ของ D



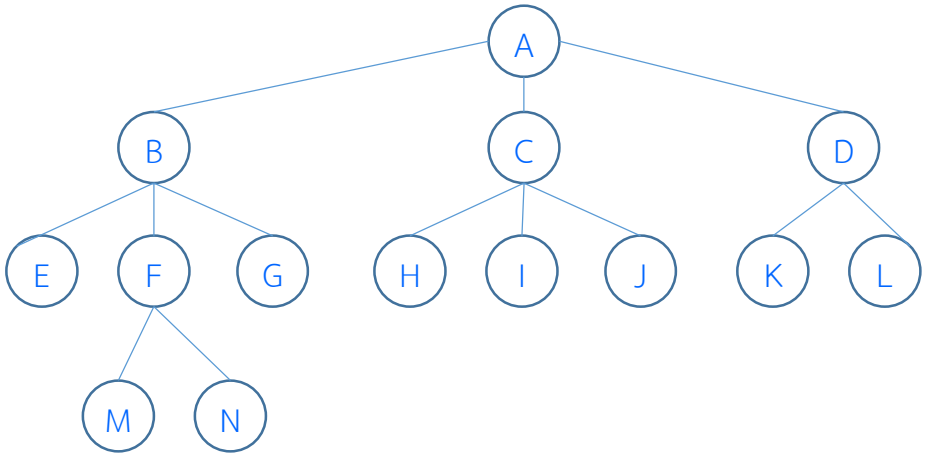
Trees:

Siblings หรือปมพี่น้อง ใช้เรียก Node ที่มีปมพ่อเดียวกัน

ตัวอย่างเช่น

E,F,G เป็น Siblings

M, N เป็น Siblings



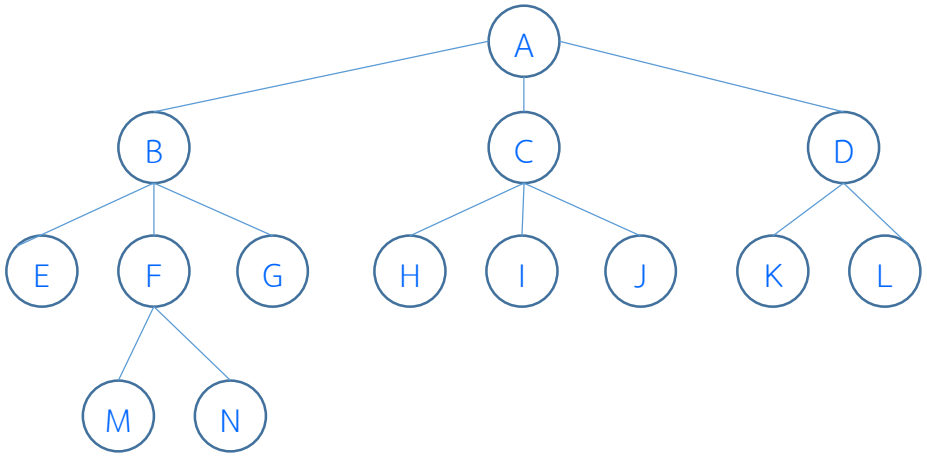
หาก Siblings มีการเรียงลำดับ ในรูปแบบใดรูปแบบหนึ่ง จะเรียกต้นไม้นี้ว่า ordered tree

Trees:

Leaf node หรือปมใบ คือปมที่ไม่มีปมลูก

ตัวอย่างเช่น

E, G, H, I, J, K, L, M, N



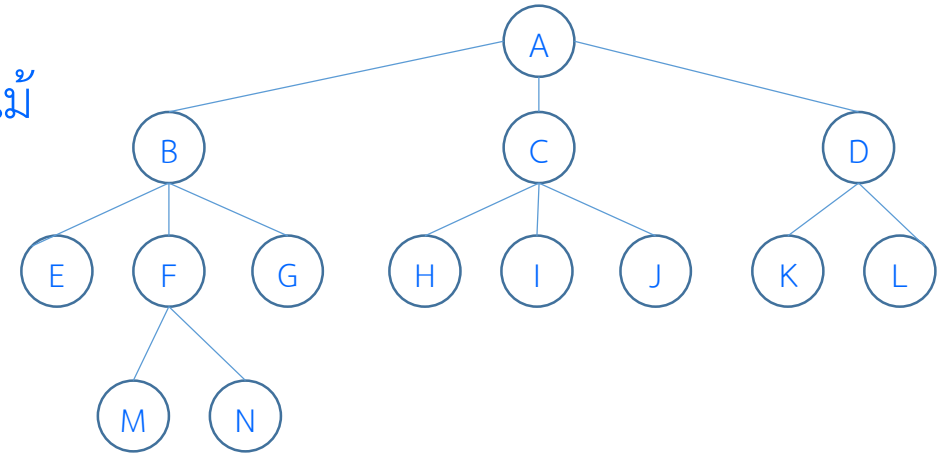
Trees:

เส้นทางจาก Root มายัง node ใด ๆ จะมีได้เพียงเส้นทางเดียวเท่านั้น

การเดินทางมายัง N จะมีได้เส้นทางเดียวเท่านั้นคือ

A, B, F, N

หากมีมากกว่า 1 เส้นทาง โครงสร้างนี้จะไม่ใช่ต้นไม้



Node ที่อยู่ระหว่างทาง ย้อนขึ้นไปจนถึง Root จะเรียกว่า Ancestors

ตัวอย่างเช่น Ancestors ของ N คือ A, B, F

Trees:

Descendent คือ Node ทั้งหมดที่อยู่ด้านล่าง

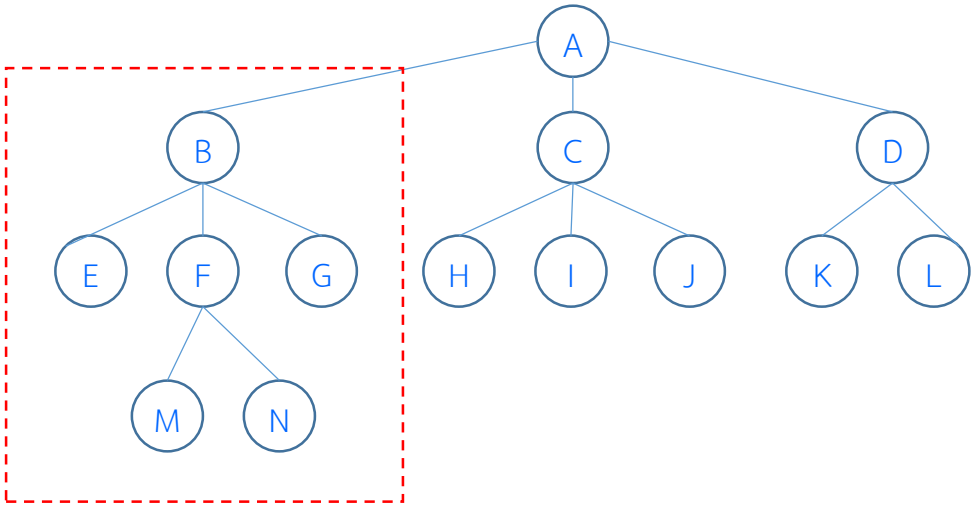
ตัวอย่างเช่น Descendants ของ B คือ

E,F,G,M,N

Descendants ยังคงโครงสร้างต้นไม้ไว้

จึงเรียกได้ว่าเป็น Subtree หรือ ต้นไม้ย่อย

โดยมี B เป็น Root



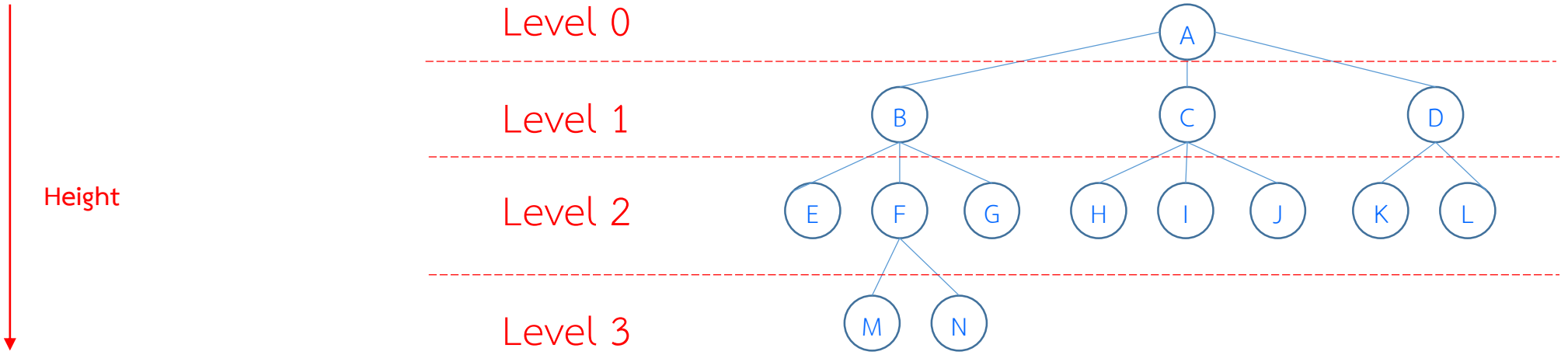
Subtree

Trees:

ระดับ หรือ ความลึก ของต้นไม้

การนับระดับของต้นไม้ จะเริ่มนับจาก Root เป็นระดับ 0

และเพิ่มระดับขึ้นเรื่อย ๆในแต่ละชั้น



จำนวนระดับสูงสุดของต้นไม้ คือความสูงของต้นไม้

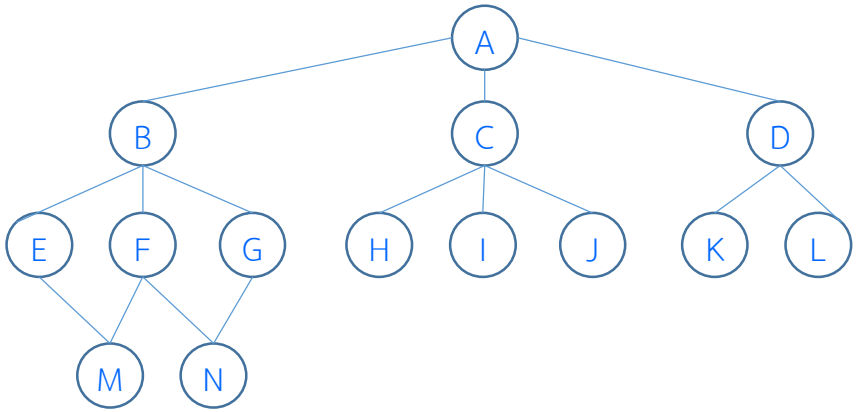
ตัวอย่างเช่นต้นไม้ต้นนี้มีความสูงเป็น 3

ต้นไม้ที่มีเพียง Root node จะมีความสูงเป็น 0

ต้นไม้ว่าง จะมีความสูงเป็น -1 หรือ NULL

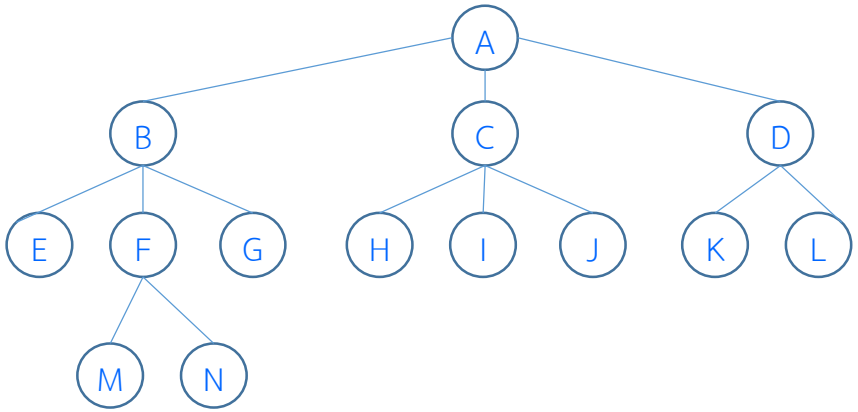
คุณสมบัติที่สำคัญของต้นไม้

- โครงสร้างไม่เป็นเชิงเส้น
- เส้นทางจาก Root ไปยัง Node ใด ๆ มีเพียงเส้นทางเดียว
- ทุก node มี parent เดียว ยกวัน root node



ไม่ใช่ต้นไม้

โครงสร้างนี้เรียกว่ากราฟ

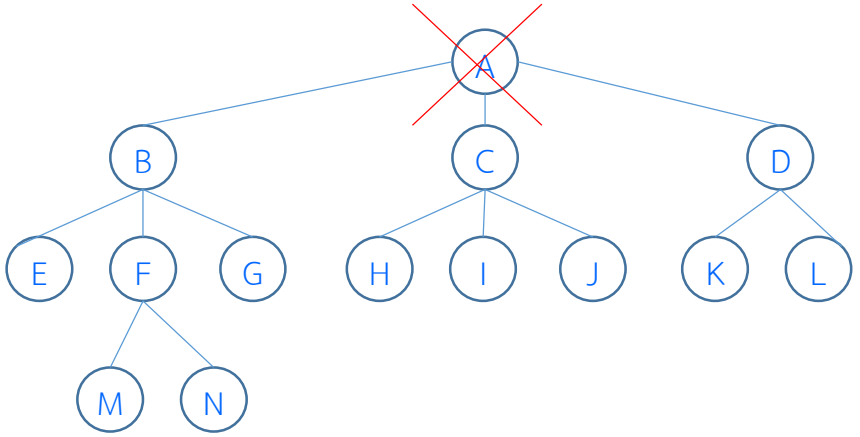


ต้นไม้

Trees:

คุณสมบัติที่สำคัญของต้นไม้

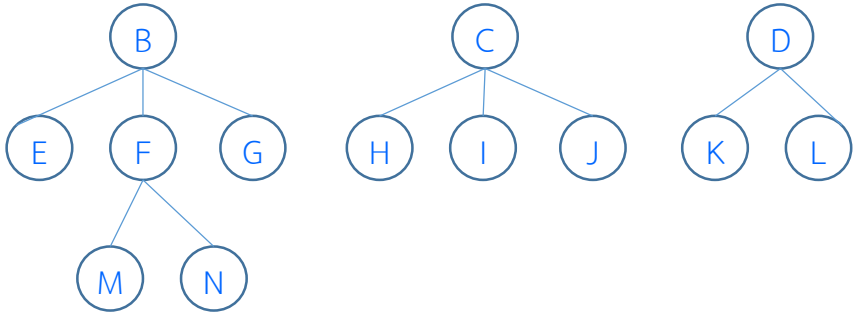
- โครงสร้างไม่เป็นเชิงเส้น
- เส้นทางจาก Root ไปยัง Node ใด ๆ มีเพียงเส้นทางเดียว
- ทุก node มี parent เดียว ยกวัน root node



Trees:

คุณสมบัติที่สำคัญของต้นไม้

- โครงสร้างไม่เป็นเชิงเส้น
- เส้นทางจาก Root ไปยัง Node ใด ๆ มีเพียงเส้นทางเดียว
- ทุก node มี parent เดียว ยกวัน root node
- ต้นไม้หลายต้นที่ไม่มี node เชื่อมโยงกัน เรียกว่า Forest

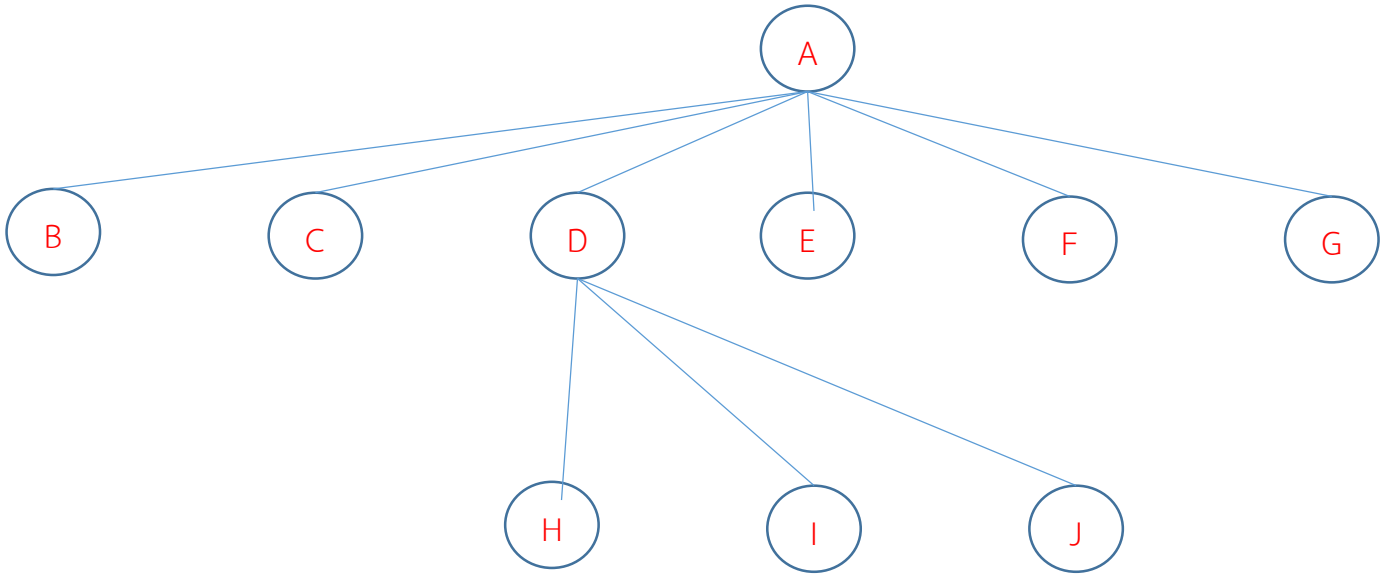


Forest

การจำแนกประเภทต้นไม้

ต้นไม้นั้นจะจำแนกประเภทจากจำนวน ลูก สูงสุดที่แต่ละ node จะมีได้

ต้นไม้ที่จำนวนลูกสูงสุด N เรียกว่า N-ary หรือ N-way tree (ต้นไม้ N ภาค)

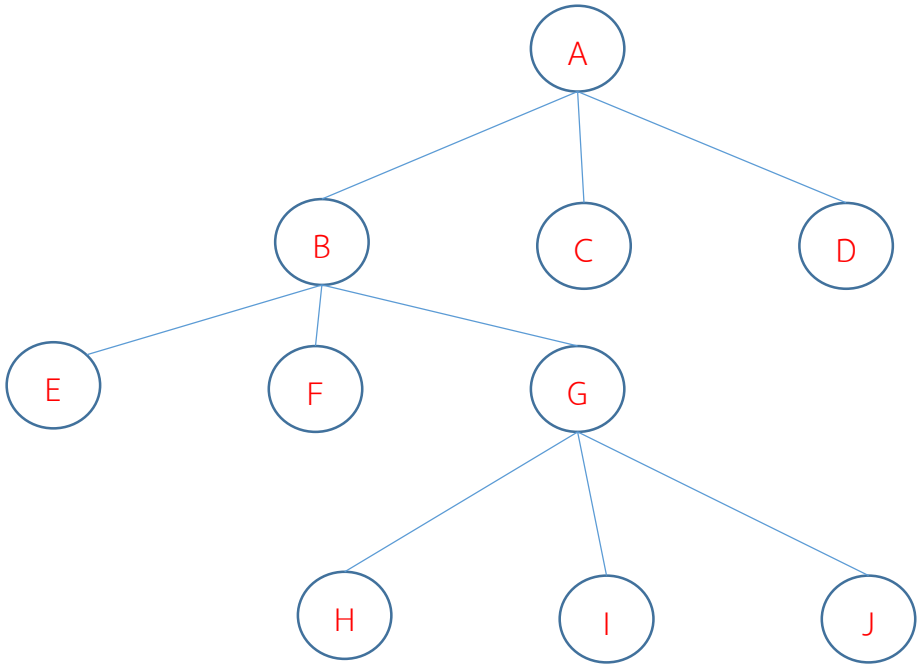


หาก node มีรูปแบบลำดับการเรียง จะเรียกว่า Multiway - tree

การจำแนกประเภทต้นไม้

ต้นไม้นั้นจะจำแนกประเภทจากจำนวน ลูก สูงสุดที่แต่ละ node จะมีได้

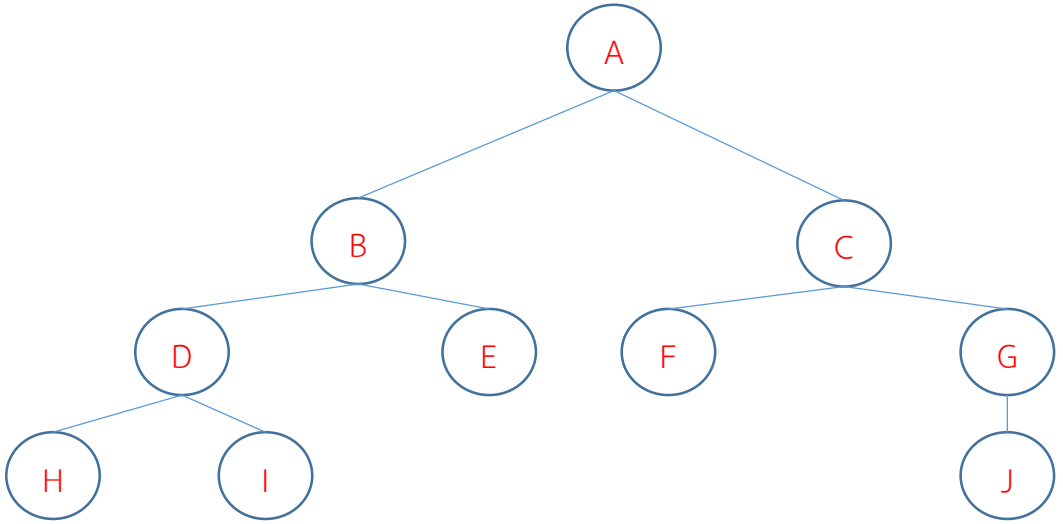
ต้นไม้ที่จำนวนลูกสูงสุด 3 เรียกว่า ternary tree



การจำแนกประเภทต้นไม้

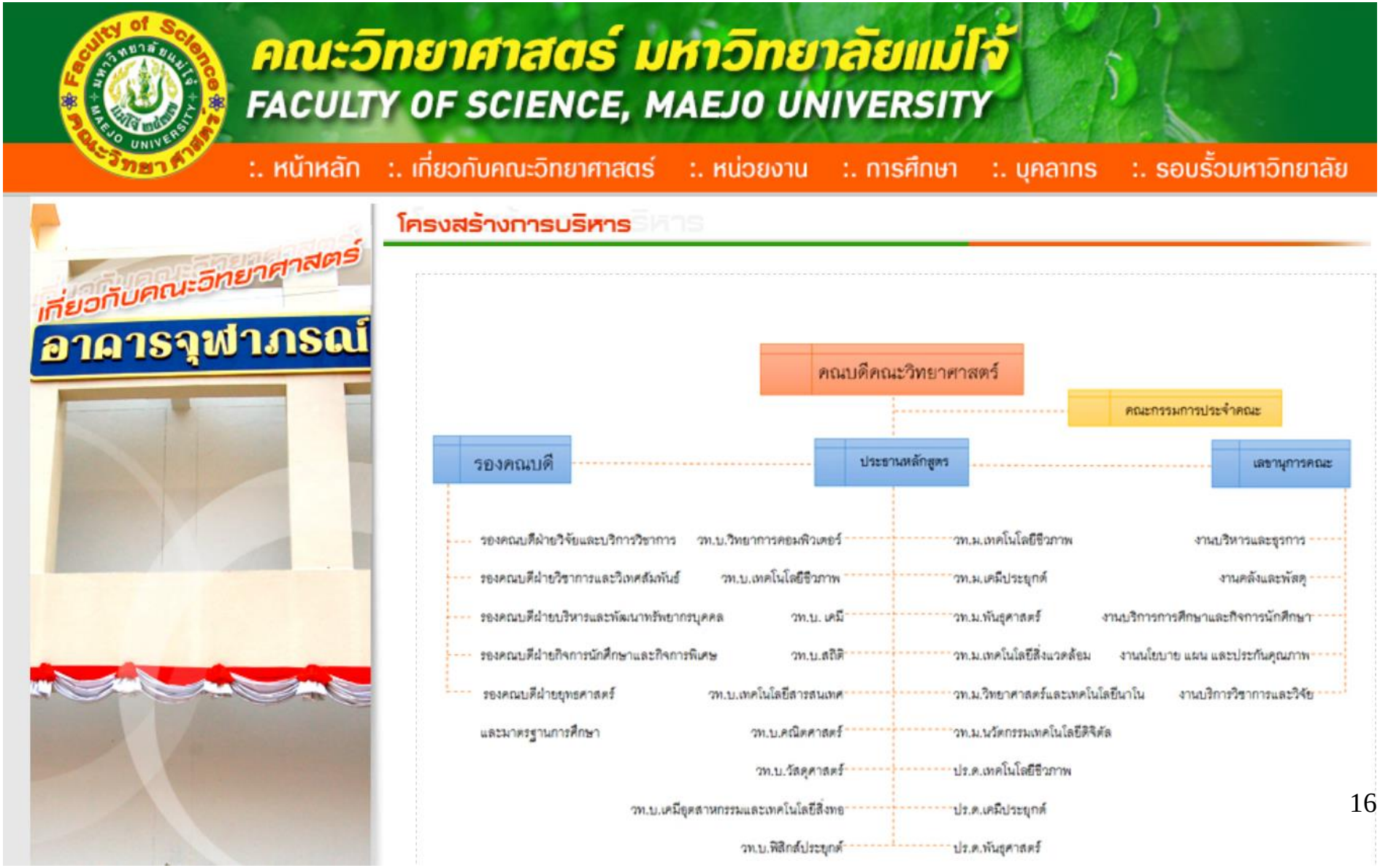
ต้นไม้นั้นจะจำแนกประเภทจากจำนวน ลูก สูงสุดที่แต่ละ node จะมีได้

ต้นไม้ที่จำนวนลูกสูงสุด 2 เรียกว่า Binary tree หรือต้นไม้ทวิภาค



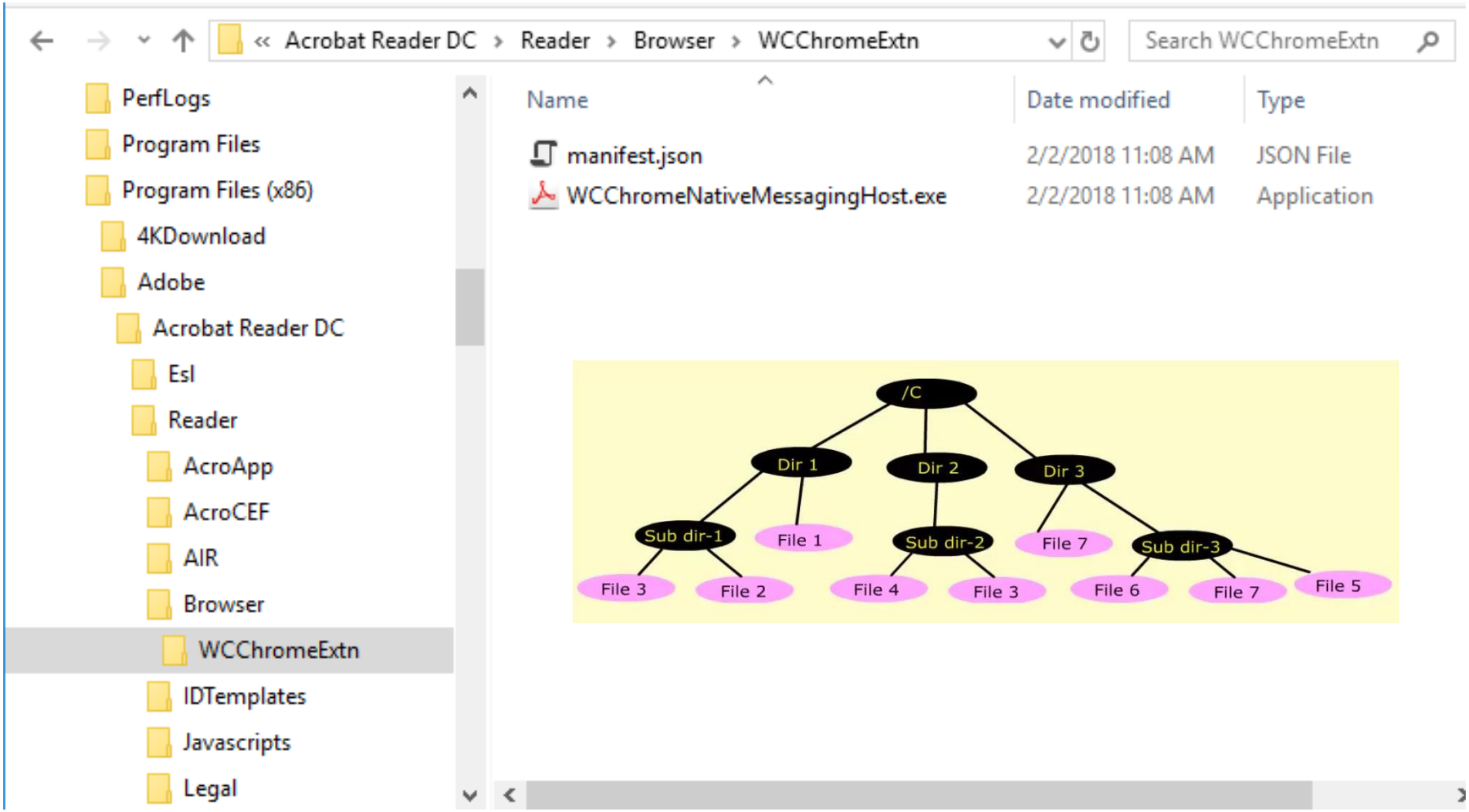
ต้นไม้ใช้ทำอะไรได้บ้าง

ฝังองค์กร



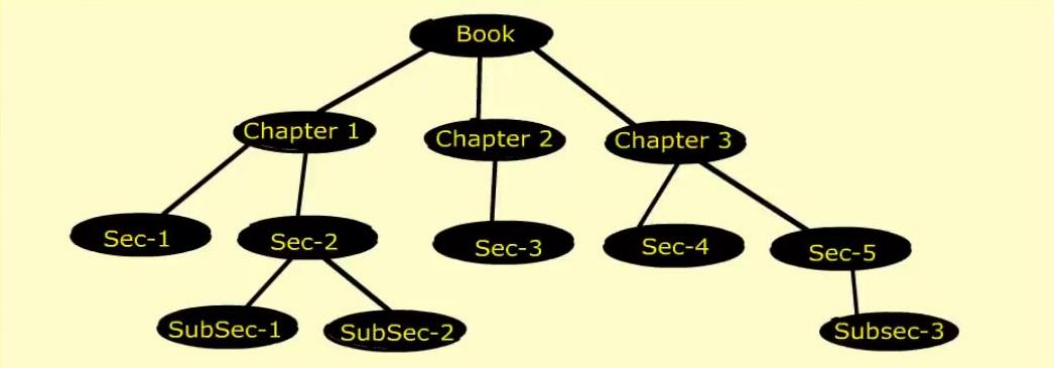
ต้นไม้ใช้ทำอะไรได้บ้าง

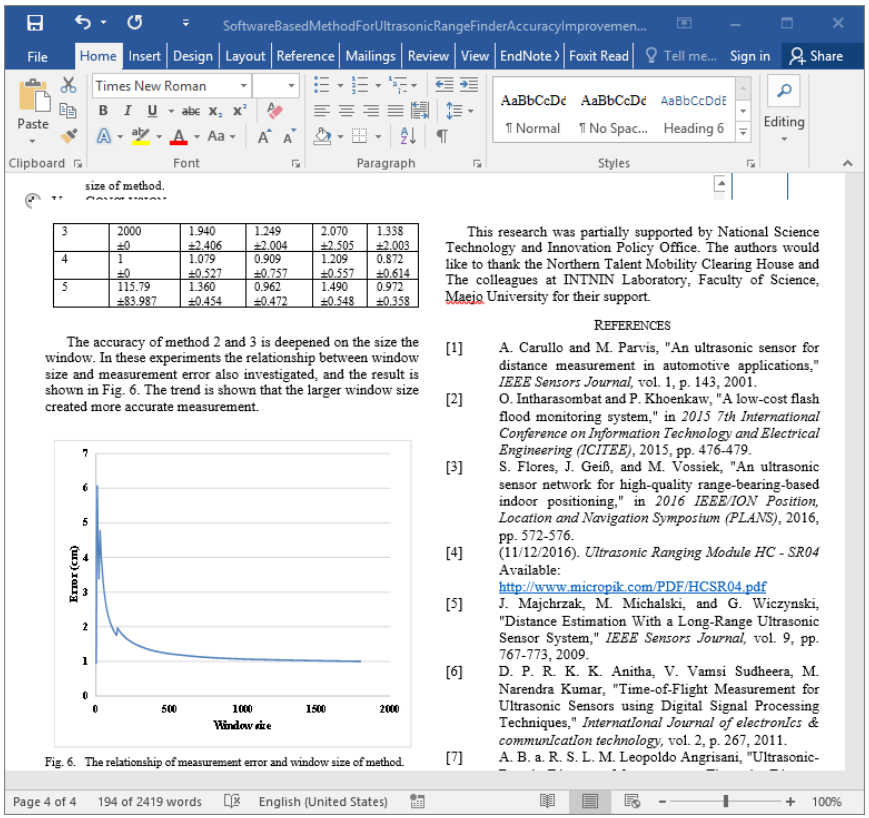
โครงสร้างแฟ้มข้อมูล

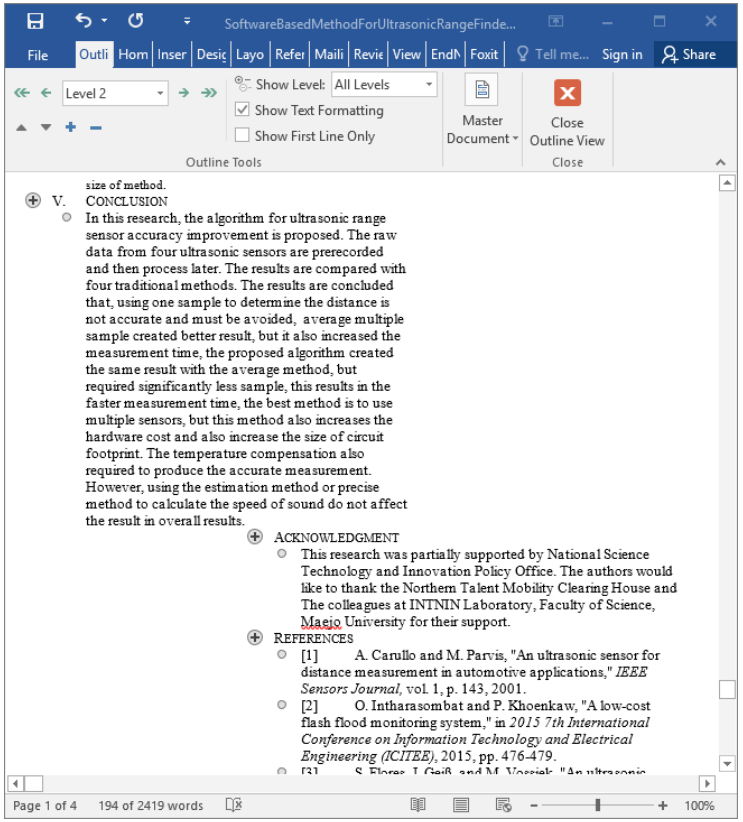


ต้นไม้ใช้ทำอะไรได้บ้าง

โครงสร้างเอกสาร

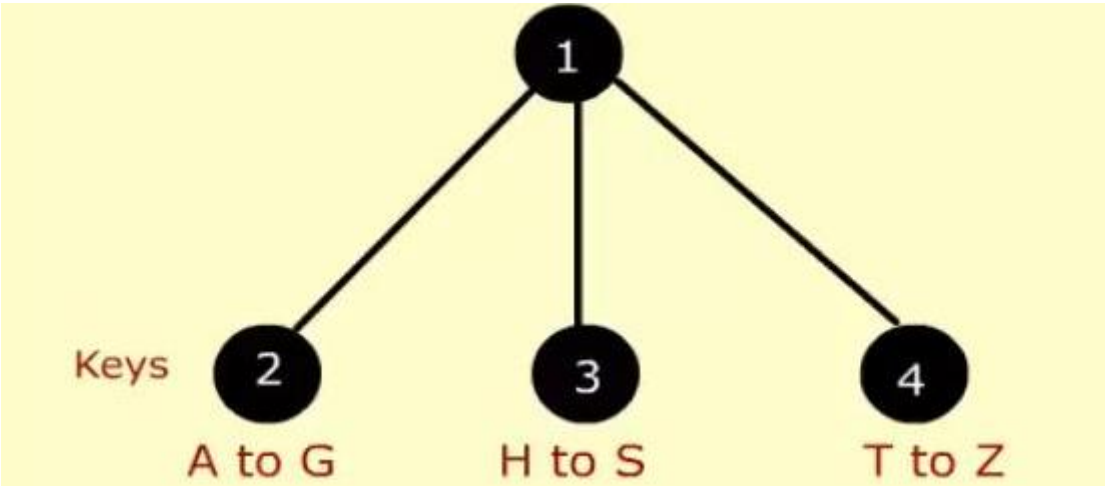






ต้นไม้ใช้ทำอะไรได้บ้าง

ฐานข้อมูล



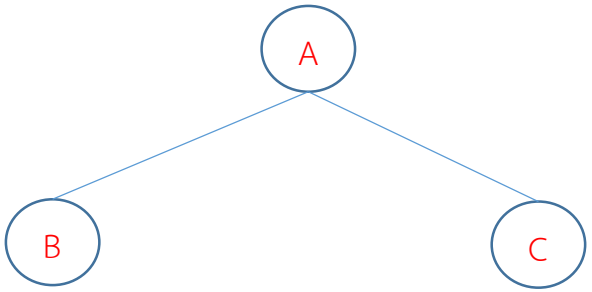
Milti-way tree ใช้ในการเก็บข้อมูลในระบบฐานข้อมูล เพื่อให้สืบค้นได้เร็ว

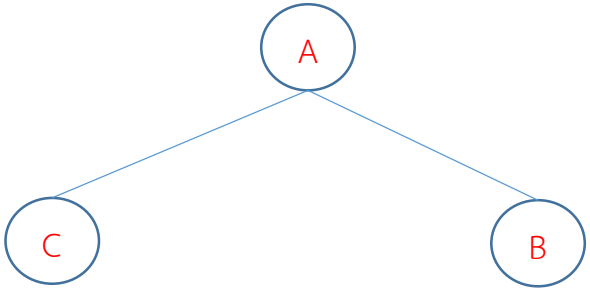
วิชานี้เราจะศึกษาต้นไม้ทวิภาคเป็นหลัก

We will focus on Binary Tree fist!

ต้นไม้ทวิภาค

- ต้นไม่ที่แต่ละ node จะมีลูกได้ไม่เกิน 2
- ลูกที่อยู่ path ด้านซ้ายเรียกว่า node ซ้าย ลูกที่อยู่ node ด้านขวาเรียกว่า nodeขวา
- การกำหนดลำดับของข้อมูลใน node ซ้ายและขวาต่างกัน จะได้ต้นไม้โครงสร้างต่างกัน



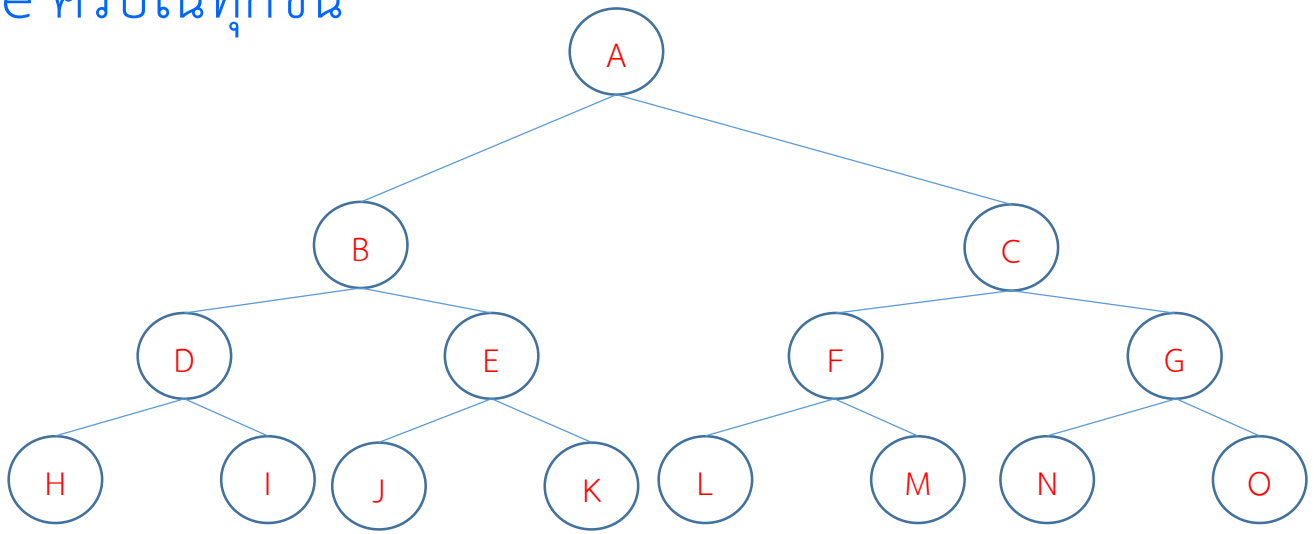


ต้นไม้ 2 ต้นนี้มีข้อมูลเมือนกัน แต่มีโครงสร้างต่างกัน

ต้นไม้ทวิภาคมีชื่อเรียกต่างกันตามคุณสมบัติของมัน

Full Binary Tree

คือต้นไม้ที่มี node ครบในทุกชั้น



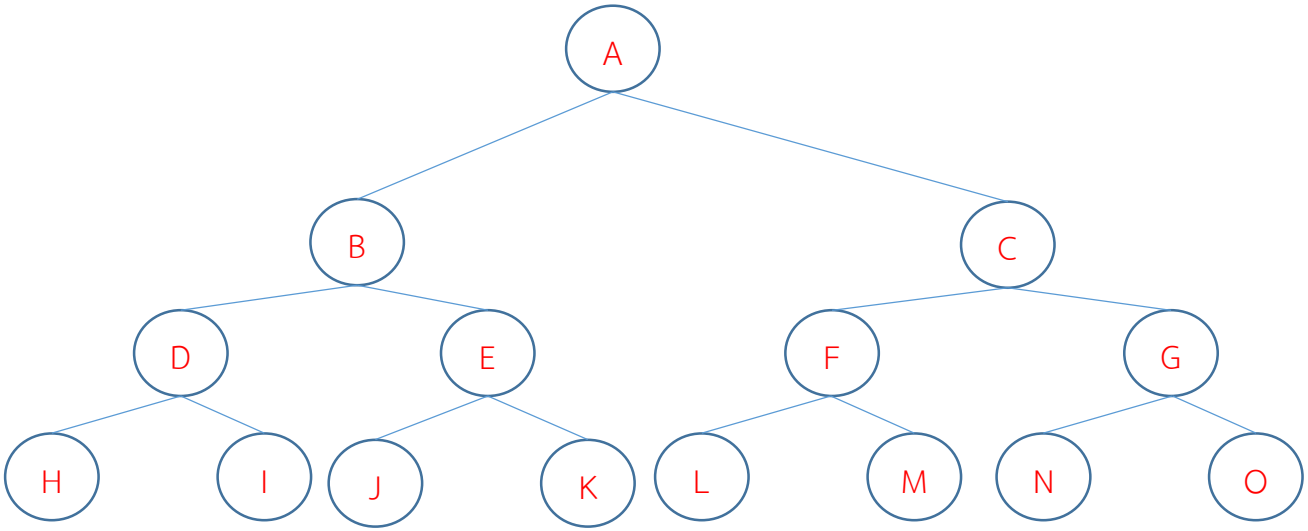
จำนวน node ในแต่ละชั้นจะสามารถคำนวณได้

$$N = 2^{h+1} - 1$$

หรือหากรู้จำนวน node ทั้งหมด ก็สามารถคำนวณความสูงได้

$$h = log_2(N + 1) - 1$$

Full Binary Tree



ต้นไม้แบบ Full binary tree ที่มีจำนวน 15 node จะมีความสูงเท่าใด

$$h = log_2(N+1) - 1$$

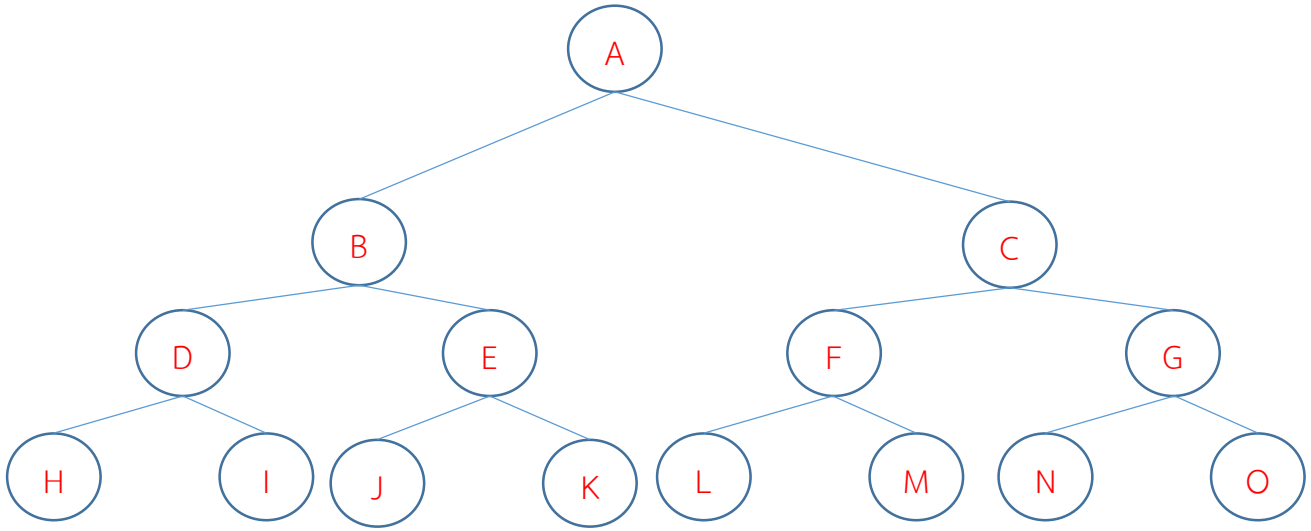
$$h = log_2(15+1) - 1$$

$$h = log_2(16) - 1$$

$$h = 4 - 1$$

$$h = 3$$

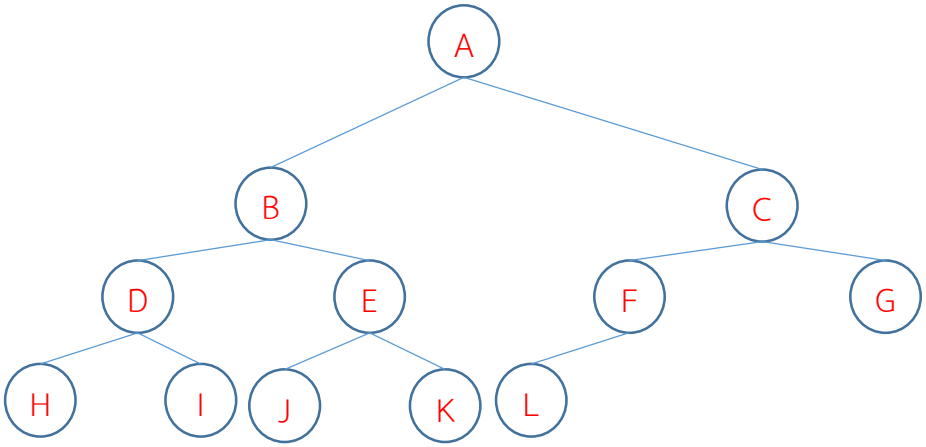
Full Binary Tree



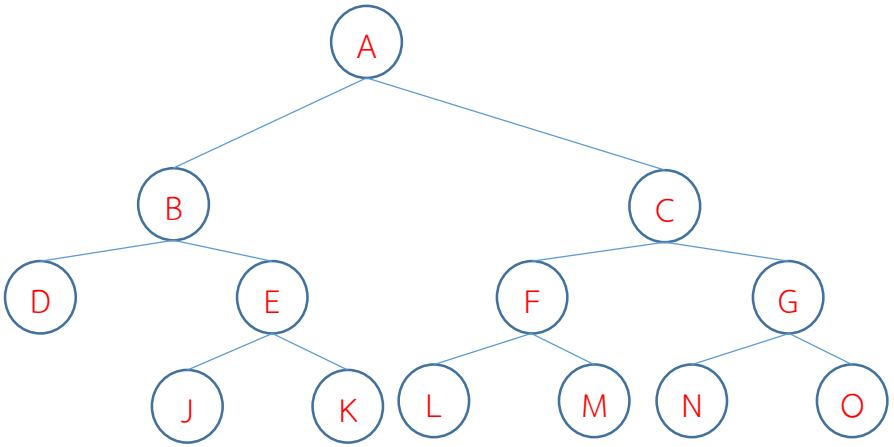
ต้นไม้แบบ Full binary tree มีโอกาสเกิดได้ยากในทางปฏิบัติ หากยอมให้ node ในชั้นล่างสุด (h-1) ไม่ครบได้ เฉพาะทางด้านขวา จะเรียกว่า

Complete Binary Tree

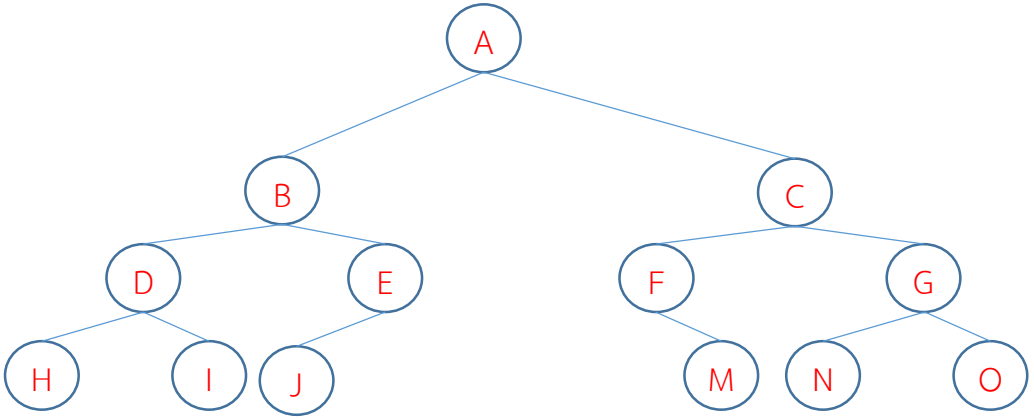
Complete Binary Tree



เป็น Complete Binary Tree



ไม่เป็น Complete Binary Tree



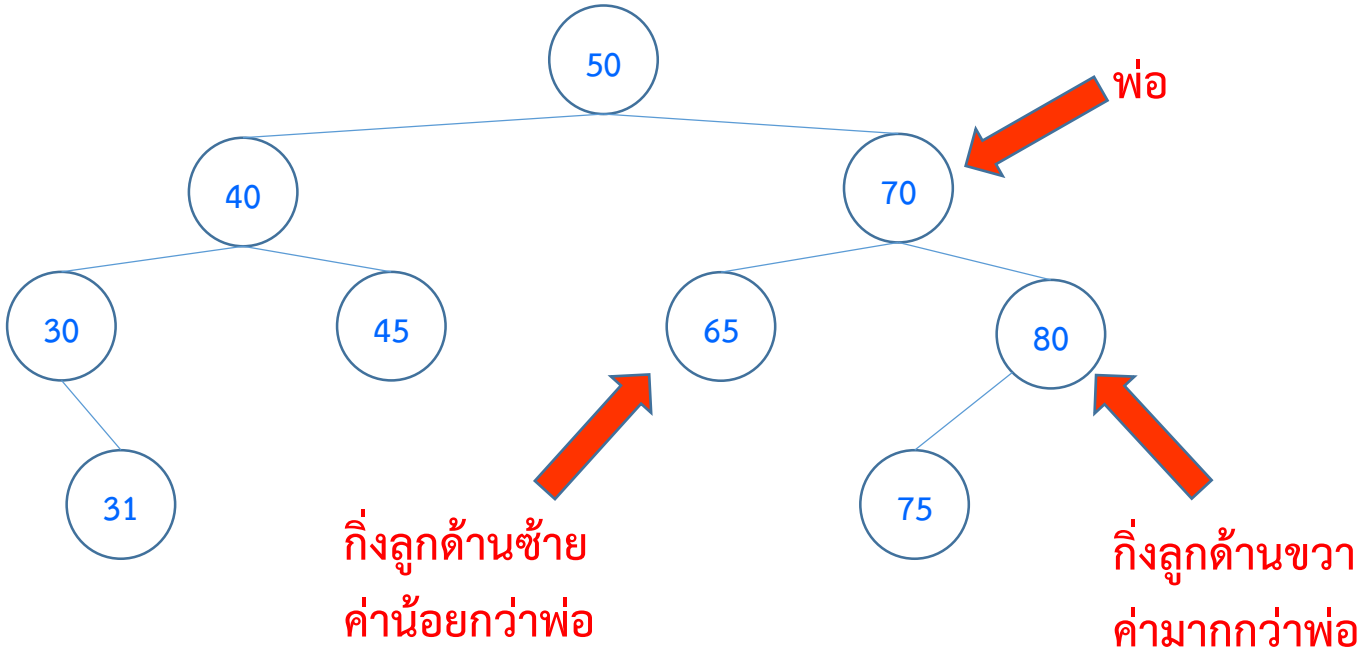
ไม่เป็น Complete Binary Tree

ต้นไม้ทวิภาคมี 4 ประเภท

1) Ordered Search Trees
2) Expression Trees
3) Heap Trees
4) Binary Decision Trees

Binary Search Trees

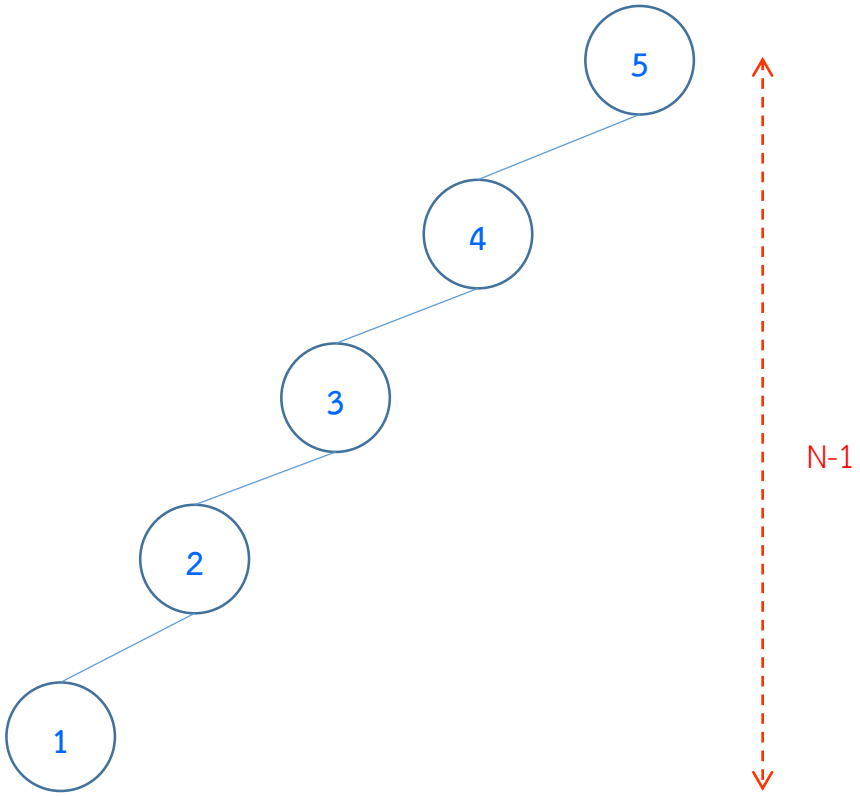
- ค่าของ node จะซ้ำกันไม่ได้
- กิ่งลูกทางด้านซ้ายจะต้องมีค่าน้อยกว่าพ่อ และกิ่งลูกทางด้านขวาต้องมีค่ามากกว่าพ่อเสมอ
- ข้อมูลในต้นไม้ต้องเป็นแบบที่สามารถเปรียบเทียบกันได้เท่านั้น



ข้อมูลไม่ซ้ำกัน

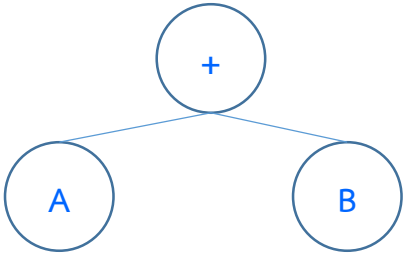
Binary Search Trees

ต้นไม้แบบ Binary Search Trees จะสูงที่สุดไม่เกินเท่าใด ?

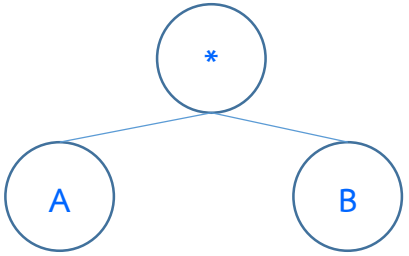


Expression Trees

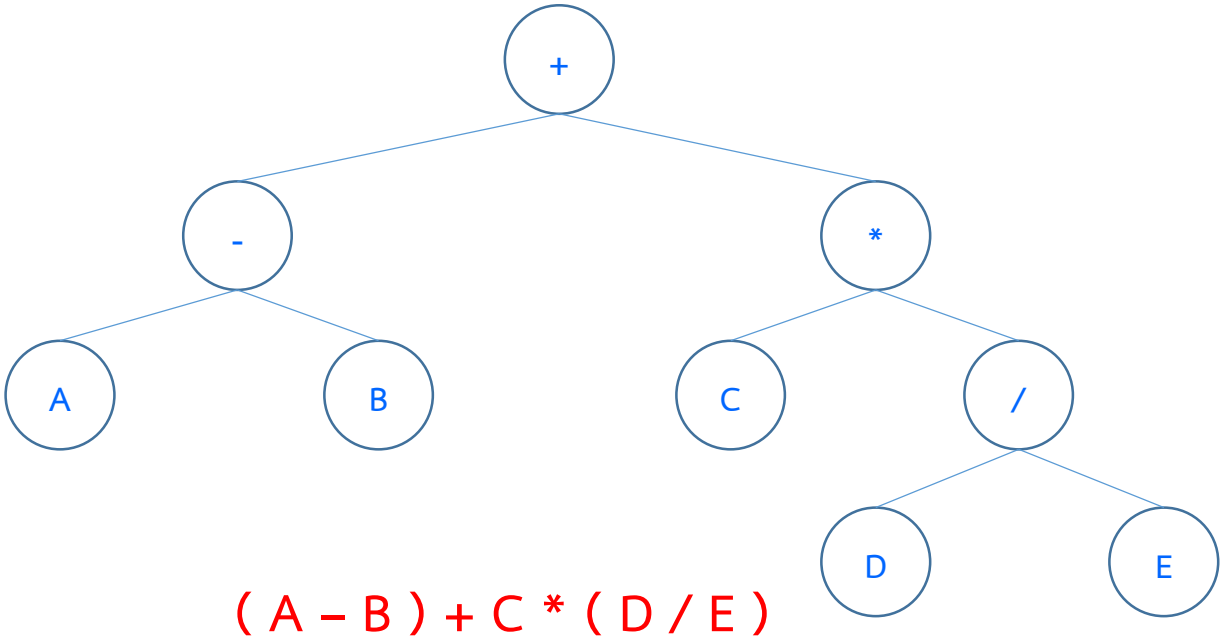
- ใช้เก็บการประมวลผลทางคณิตศาสตร์ที่มีลำดับชั้น
- Operand จะอยู่ที่ปมใบเสมอ



A + B



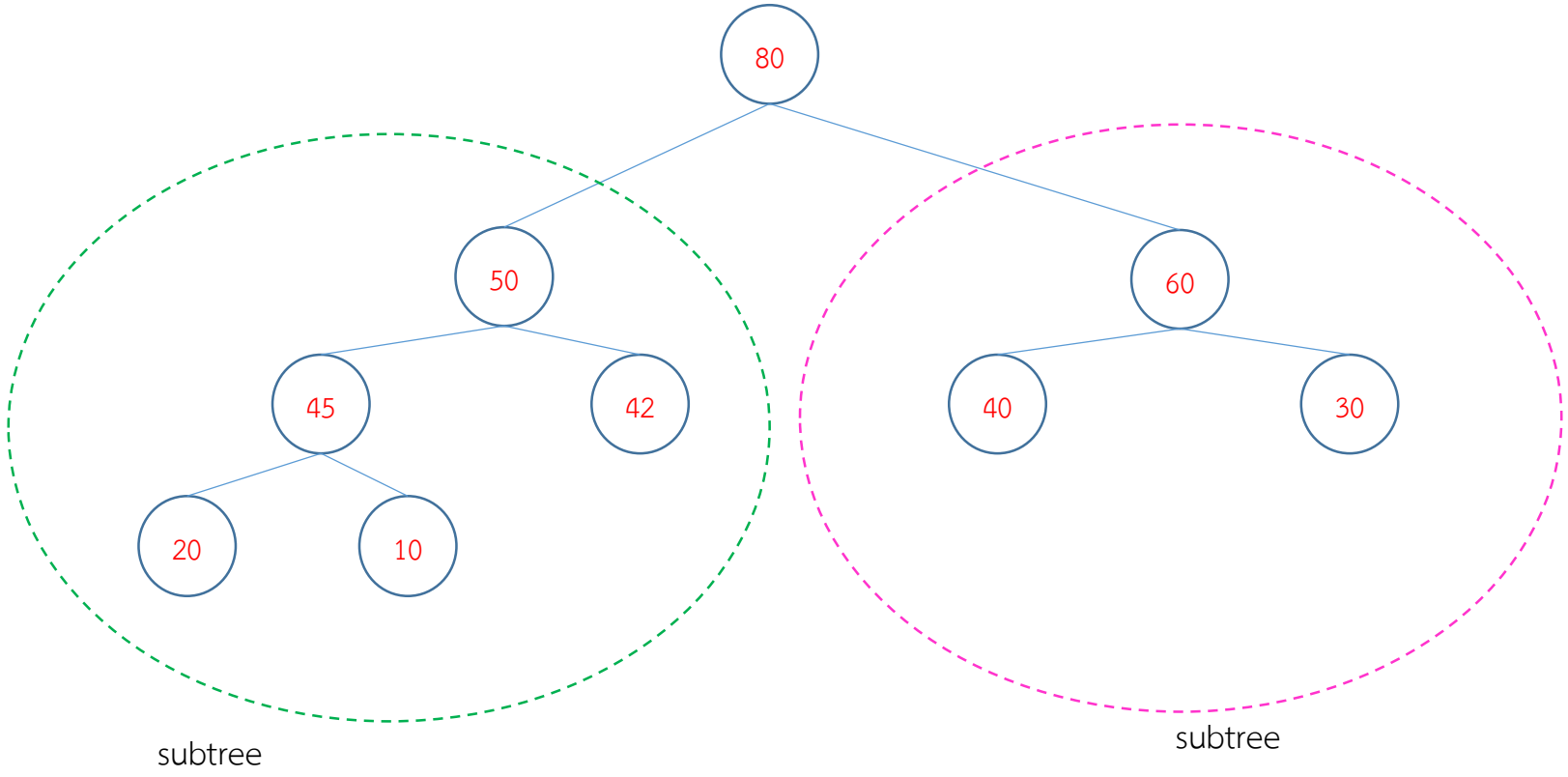
A * B



(A – B) + C * (D / E)

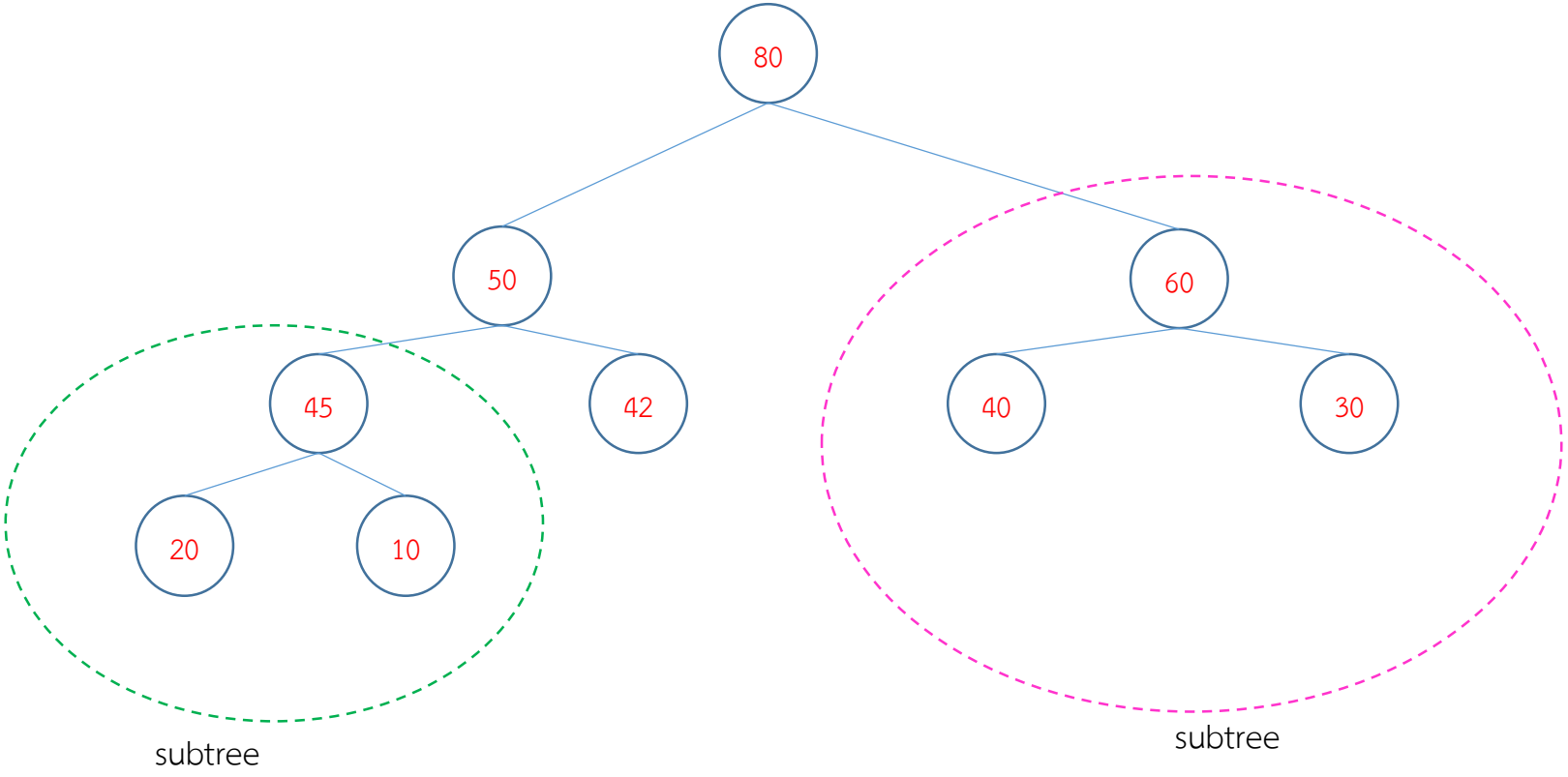
Heap Trees

- คือ Binary tree แบบพิเศษ
- ค่าของ node ใด ๆ จะมากกว่าค่าใน subtree ทางด้านซ้ายและขวา เสมอ



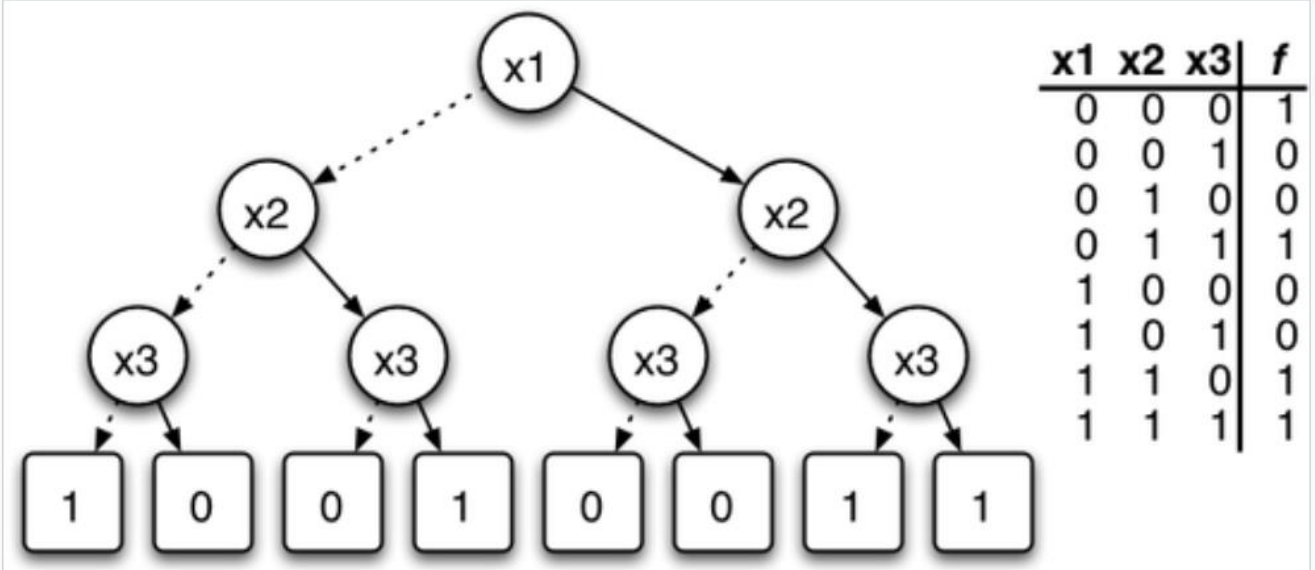
Heap Trees

- คือ Binary tree แบบพิเศษ
- ค่าของ node ใด ๆ จะมากกว่าค่าใน subtree ทางด้านซ้ายและขวา เสมอ
- นิยมใช้สร้างอัลกอริทึมเรียงลำดับข้อมูล
- ใช้สร้าง Priority Queue



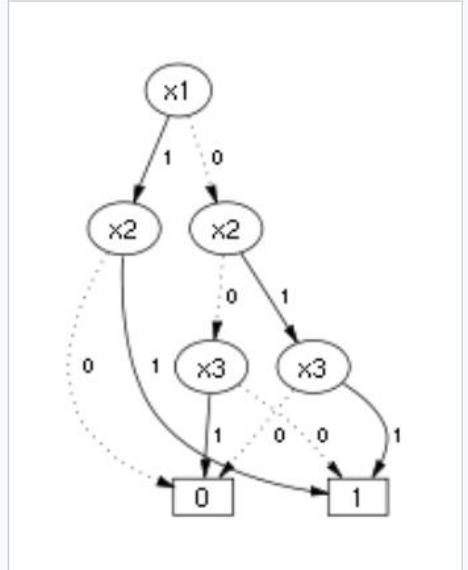
Binary Decision Trees

- ใช้สำหรับแทนลำดับของ if-then-else
- แต่ละ node จะแทนเงื่อนไขของการตัดสินใจ
- ปมใบจะแทน action ของการตัดสินใจ
- นิยมใช้ในการแปลภาษาโปรแกรมระดับสูง
- ใช้ในงานทางด้านปัญญาประดิษฐ์



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Binary decision tree and truth table for the function $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2\bar{x}_3 + x_1x_2 + x_2x_3$



BDD for the function f

https://en.wikipedia.org/wiki/Binary_decision_diagram

การสร้างต้นไม้

สิ่งที่ต้องคำนึง

- การเก็บโครงสร้างของต้นไม้ (ข้อมูล และ linkของกิ่ง)
  - ใช้ Struct
  - ใช้ Array หลายตัว
- การจัดการข้อมูลของภายในต้นไม้
  - Add
  - Remove
  - ....
- วิธีการเก็บข้อมูล
  - ใช้ Array
  - ใช้ Linked-list

วิธีที่ 1 ใช้ array ของ structure

สิ่งที่ต้องคำนึง

- การเก็บโครงสร้างของต้นไม้ (ข้อมูล และ linkของกิ่ง)
  - ใช้ Struct
  - ใช้ Array หลายตัว
- การจัดการข้อมูลของภายในต้นไม้
  - Add
  - Remove
  - ....
- วิธีการเก็บข้อมูล
  - ใช้ Array
  - ใช้ Linked-list

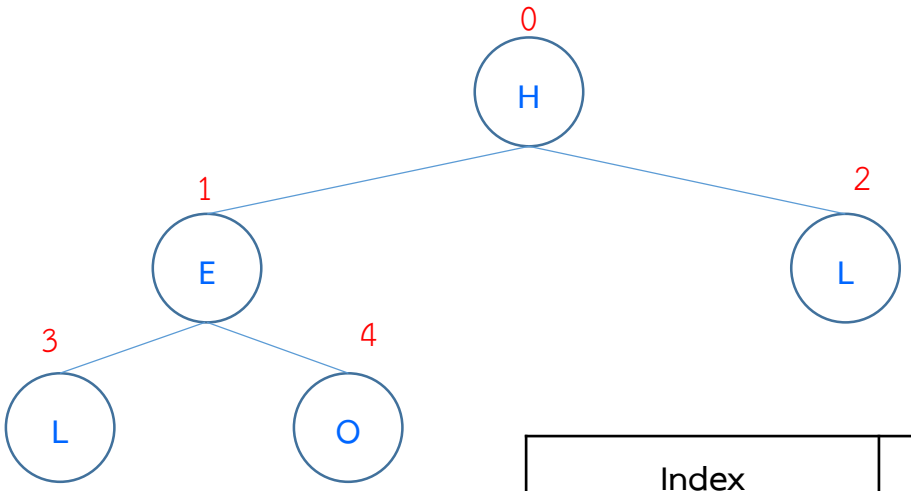
วิธีที่ 1 ใช้ array ของ structure

ข้อมูล	Link ไปยังลูกด้านซ้าย	Link ไปยังลูกด้านขวา

Struct

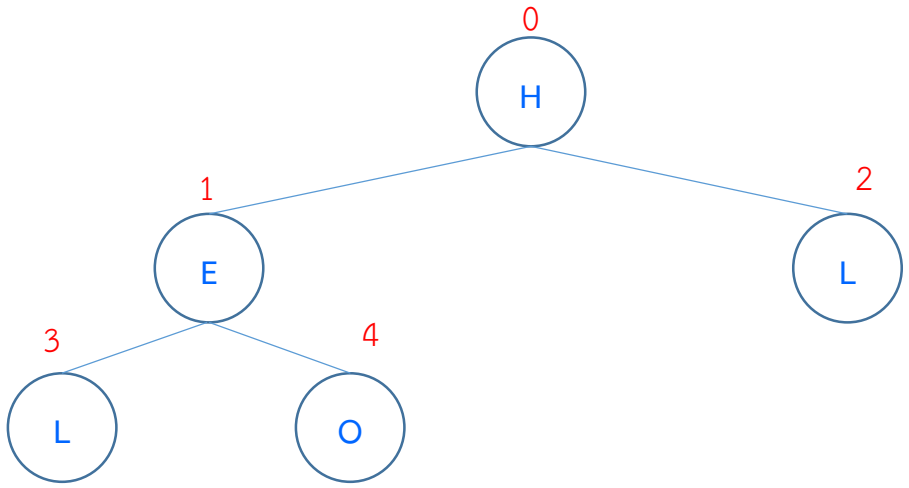
```c
#include<stdio.h>
struct Node{
    char c;
    int left;
    int right;
};

main(){
    struct Node node[100];
}
```

วิธีที่ 1 ใช้ array ของ structure



Index	Data	Left	Right
0	H	1	2
1	E	3	4
2	L	X	X
3	L	X	X
4	O	X	X

วิธีที่ 1 ใช้ array ของ structure



Index	Data	Left	Right
0	H	1	2
1	E	3	4
2	L	X	X
3	L	X	X
4	O	X	X

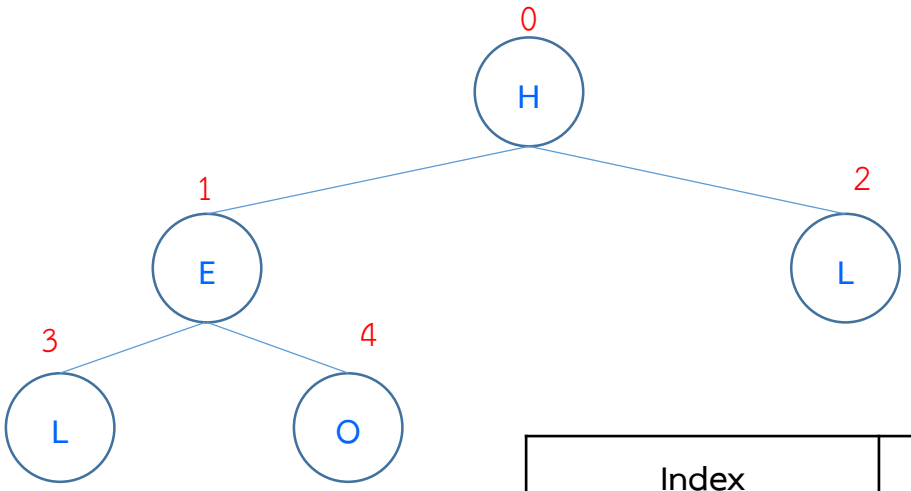
```c
#include<stdio.h>
struct Node{
    char c;
    int left;
    int right;
};

main(){
    struct Node node[100];
    node[0].c='H';
    node[0].left=1;
    node[0].right=2;

    node[1].c='E';
    node[1].left=3;
    node[1].right=4;

    node[2].c='L';
    node[2].left=0;
    node[2].right=0;

    node[3].c='L';
    node[3].left=0;
    node[3].right=0;

    node[4].c='O';
    node[4].left=0;
    node[4].right=0;

}
```

วิธีที่ 2 ใช้ array อย่างเดียว

สิ่งที่ต้องคำนึง

- การเก็บโครงสร้างของต้นไม้ (ข้อมูล และ linkของกิ่ง)
  - ใช้ Struct
  - ใช้ Array หลายตัว
- การจัดการข้อมูลของภายในต้นไม้
  - Add
  - Remove
  - ....
- วิธีการเก็บข้อมูล
  - ใช้ Array
  - ใช้ Linked-list

วิธีที่ 2 ใช้ array อย่างเดียว




```c
#include<stdio.h>

main(){
    char Data[]={'H','E','L','L','O'};
    int Left[]={1,3,0,0,0};
    int Right[]={2,4,0,0,0};
}
```

Index	Data	Left	Right
0	H	1	2
1	E	3	4
2	L	X	X
3	L	X	X
4	O	X	X

Array ตัวที่ 1 (Data)

Array ตัวที่ 2 (Left)

Array ตัวที่ 3 (Right)

การลบหรือแทรก ทำได้ยาก เพราะต้องแก้ข้อมูลใน Array ถึง 3 ตัว

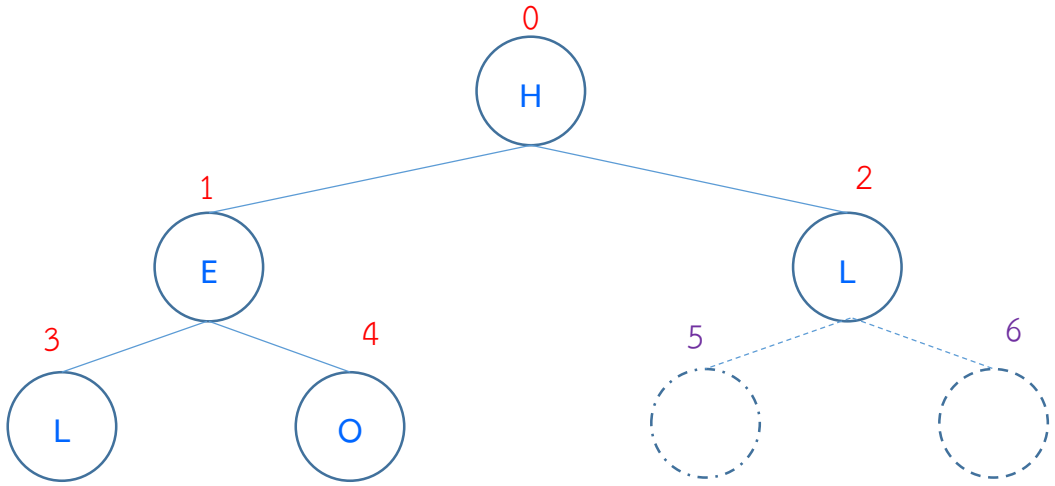
วิธีที่ 3 ใช้ array ตัวเดียว

สิ่งที่ต้องคำนึง

- การเก็บโครงสร้างของต้นไม้ (ข้อมูล และ linkของกิ่ง)
  - ใช้ Struct
  - ใช้ Array หลายตัว
- การจัดการข้อมูลของภายในต้นไม้
  - Add
  - Remove
  - ....
- วิธีการเก็บข้อมูล
  - ใช้ Array
  - ใช้ Linked-list

วิธีที่ 3 ใช้ array ตัวเดียว

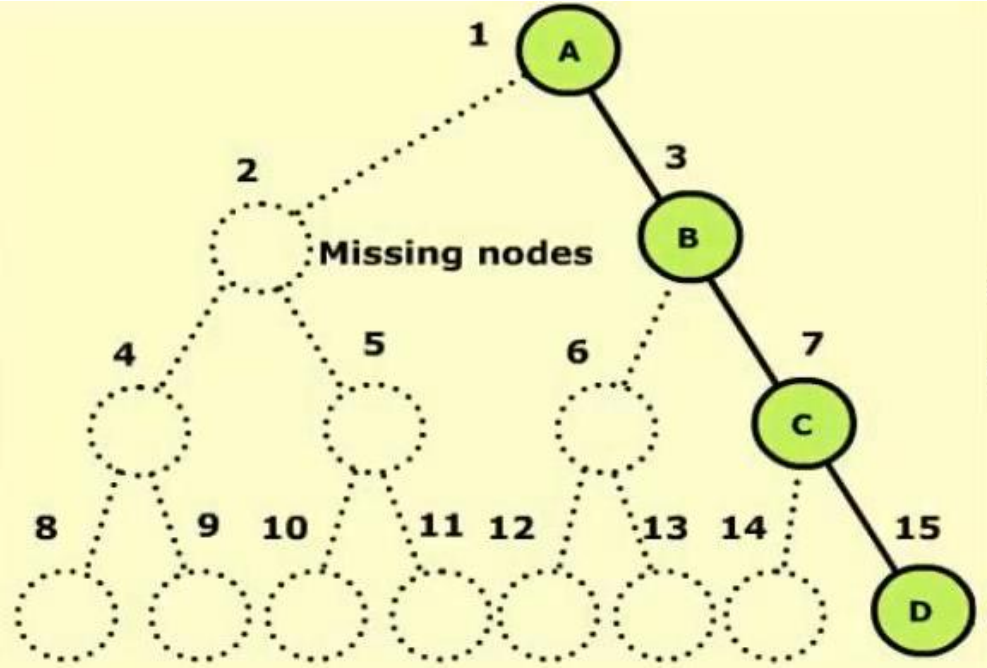
- วิธีนี้จะมองว่าต้นไม้เป็นแบบ Full binary tree
- หาก Node ไหนหายไปก็จะให้ node นั้นเป็น Null



	0	1	2	3	4	5	6
Data	H	E	L	L	O	-	-

วิธีที่ 3 ใช้ array ตัวเดียว

- วิธีนี้จะมองว่าต้นไม้เป็นแบบ Full binary tree
- หาก Node ไหนหายไปก็จะให้ node นั้นเป็น Null
- เขียนโปรแกรมง่าย
- แต่หากต้นไม้ลู่ไปทางทิศทางเดียว จะเปลืองหน่วยความจำ



Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Node	A	-	B	-	-	-	C	-	-	-	-	-	-	-	D

ข้อดีของการใช้ Array

- เข้าถึงข้อมูลใน node ได้โดยตรง
- เหมาะกับ complete tree
- เหมาะกับการค้นหาข้อมูล

ข้อเสียของการใช้ Array

- การลบข้อมูลทำได้ช้า
- สิ้นเปลืองพื้นที่หน่วยความจำ หากต้นไม้ไม่เป็น complete tree
- เปลี่ยนแปลงขนาดไม่ได้

วิธีที่ 4 ใช้ Linked-list และ Structure

สิ่งที่ต้องคำนึง

- การเก็บโครงสร้างของต้นไม้ (ข้อมูล และ linkของกิ่ง)
  - ใช้ Struct
  - ใช้ Array หลายตัว
- การจัดการข้อมูลของภายในต้นไม้
  - Add
  - Remove
  - ....
- วิธีการเก็บข้อมูล
  - ใช้ Array
  - ใช้ Linked-list

วิธีที่ 4 ใช้ Linked-list และ Structure

ข้อมูล	*Link ไปยังลูกด้านซ้าย	*Link ไปยังลูกด้านขวา

Struct

```
#include<stdio.h>
typedef struct Node{
    char c;
    struct Node *left;
    struct Node *right;
};
main(){

}
```

ข้อแตกต่างจากการใช้ Array คือ

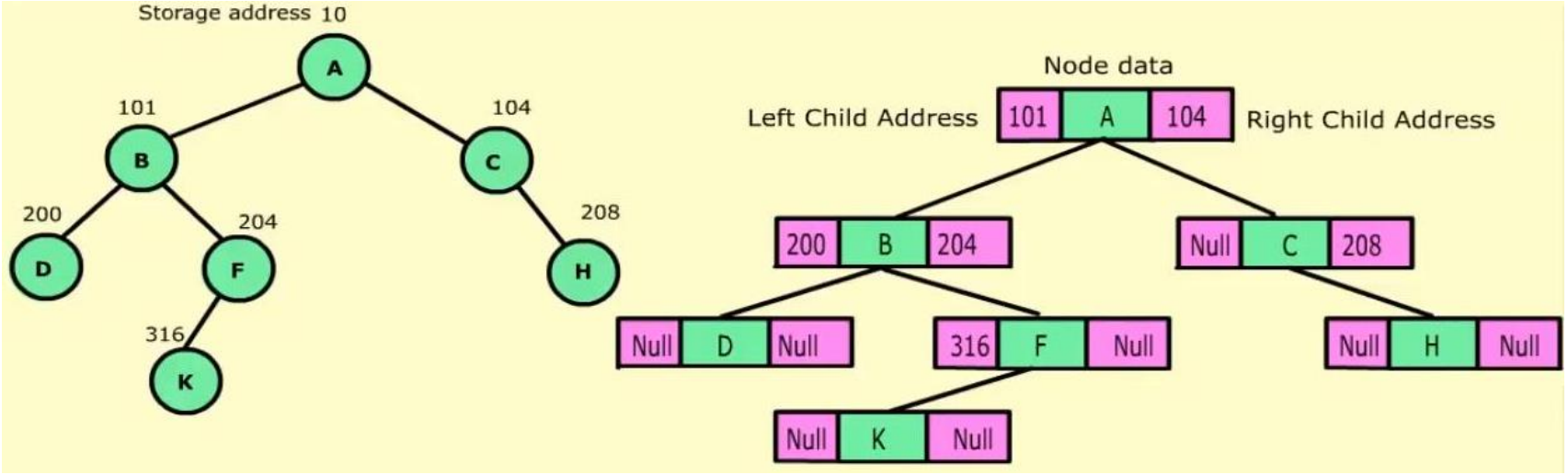
ตำแหน่ง left และ right ไม่ได้ใช้ index ของ node แต่เป็น pointer ชี้ตำแหน่งของ node ในหน่วยความจำ

วิธีที่ 4 ใช้ Linked-list และ Structure

ข้อมูล	*Link ไปยังลูกด้านซ้าย	*Link ไปยังลูกด้านขวา

Struct

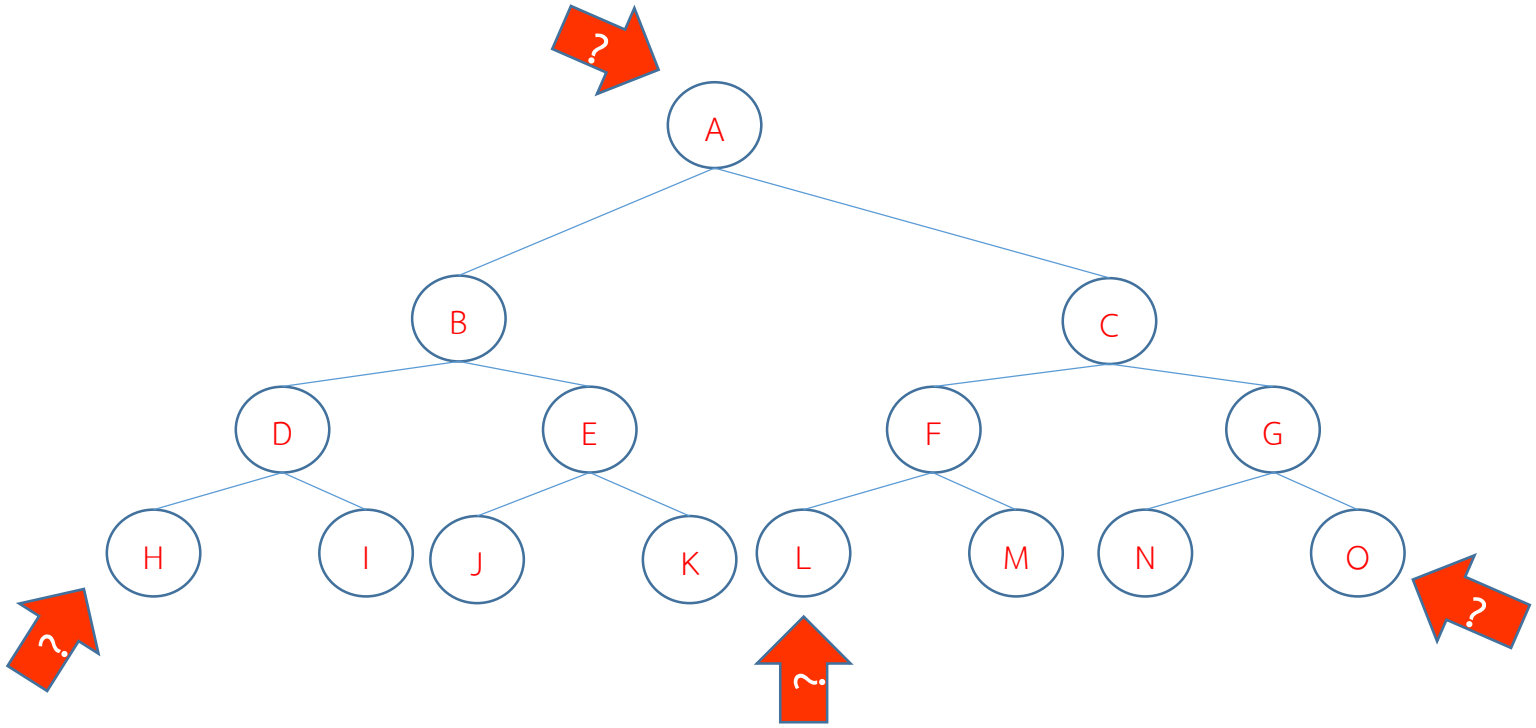
node ไหนไม่มีลูกก็กำหนดให้ link เป็น NULL



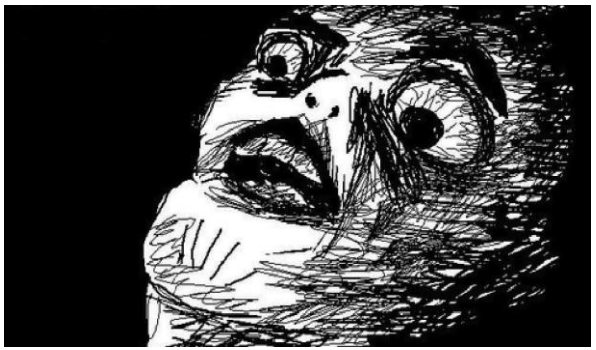
Linked-list ไม่สามารถเข้าถึงข้อมูลได้โดยตรงเหมือน Array

แก้ไขโครงสร้างจึงต้องใช้การ Traverse จาก Root

การท่องเข้าไปในต้นไม้
จะเริ่มต้นจาก node ไหน ?

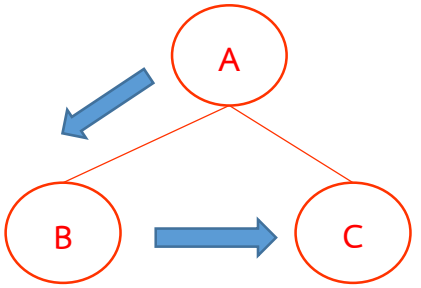


1) Preorder Traversal
2) Inorder Traversal
3) Postorder Traversal



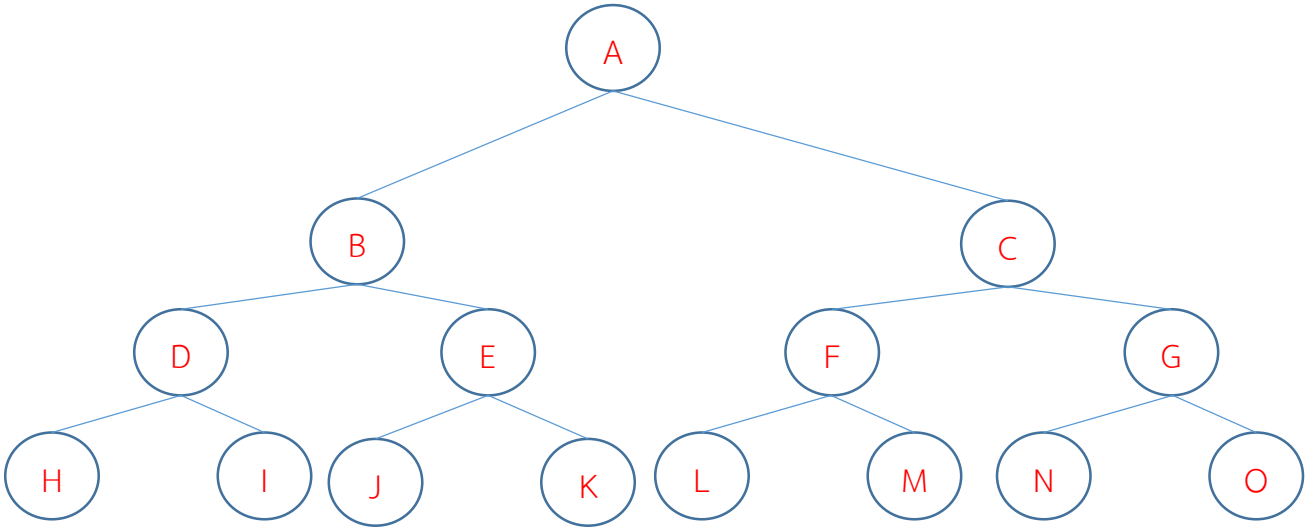
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



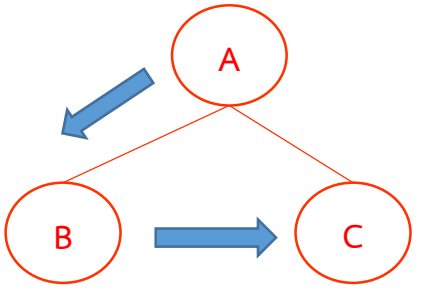
การท่องแบบ Preorder

A --> B --> C



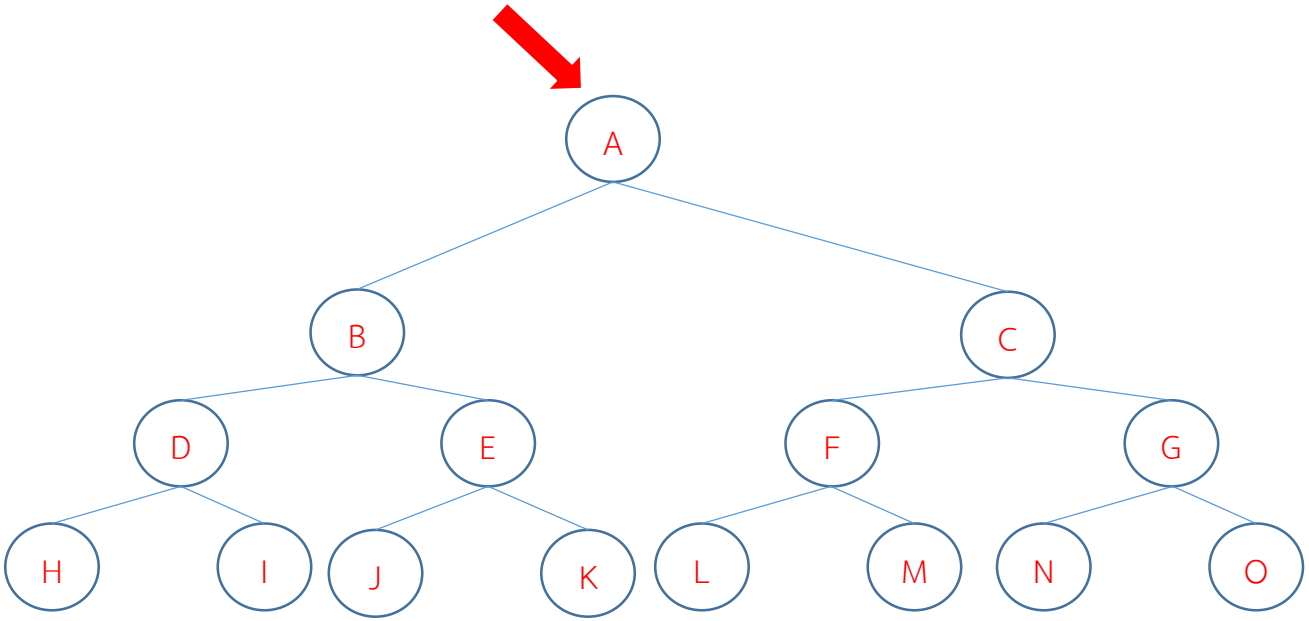
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

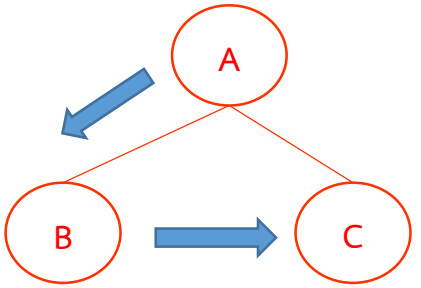
A --> B --> C



A

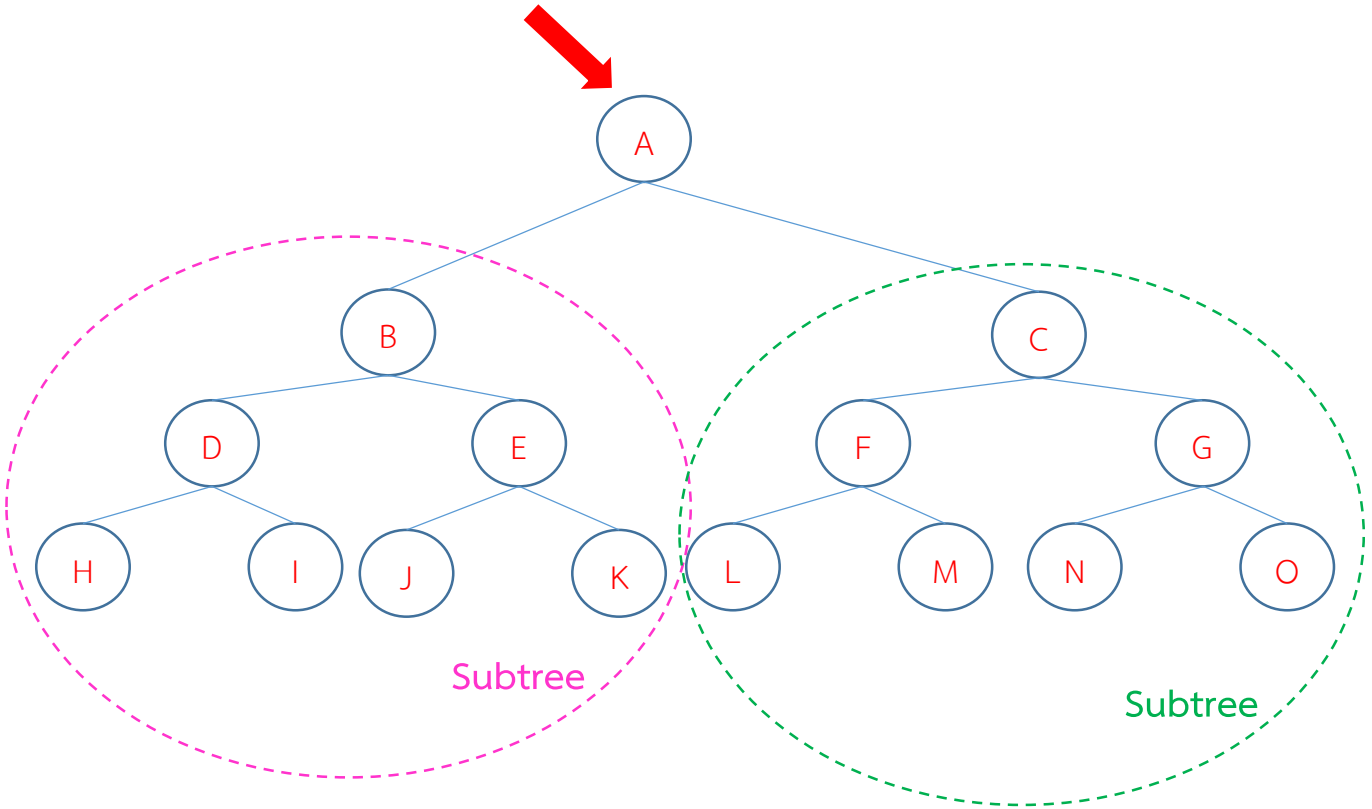
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

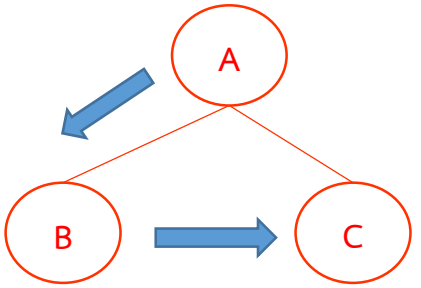
A --> B --> C



A

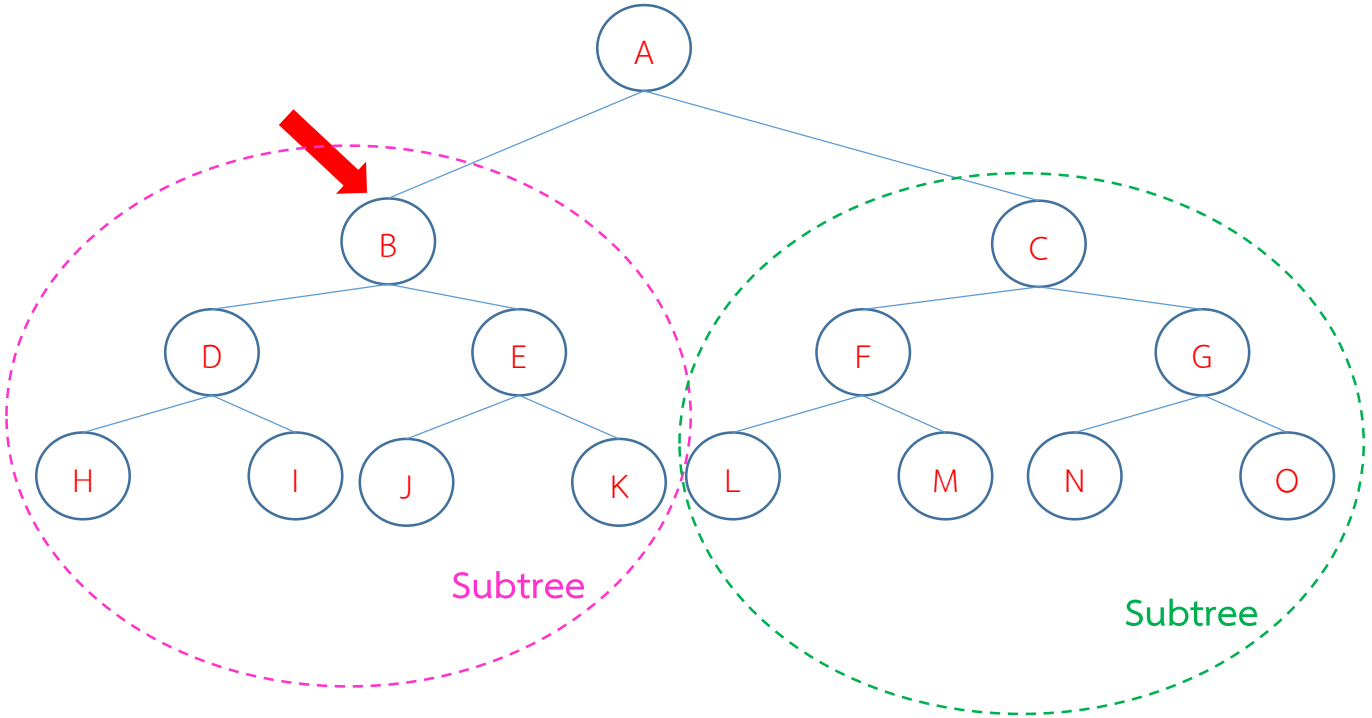
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

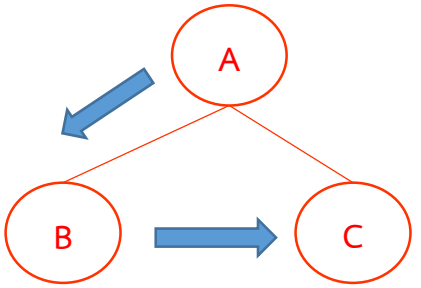
A --> B --> C



A,B

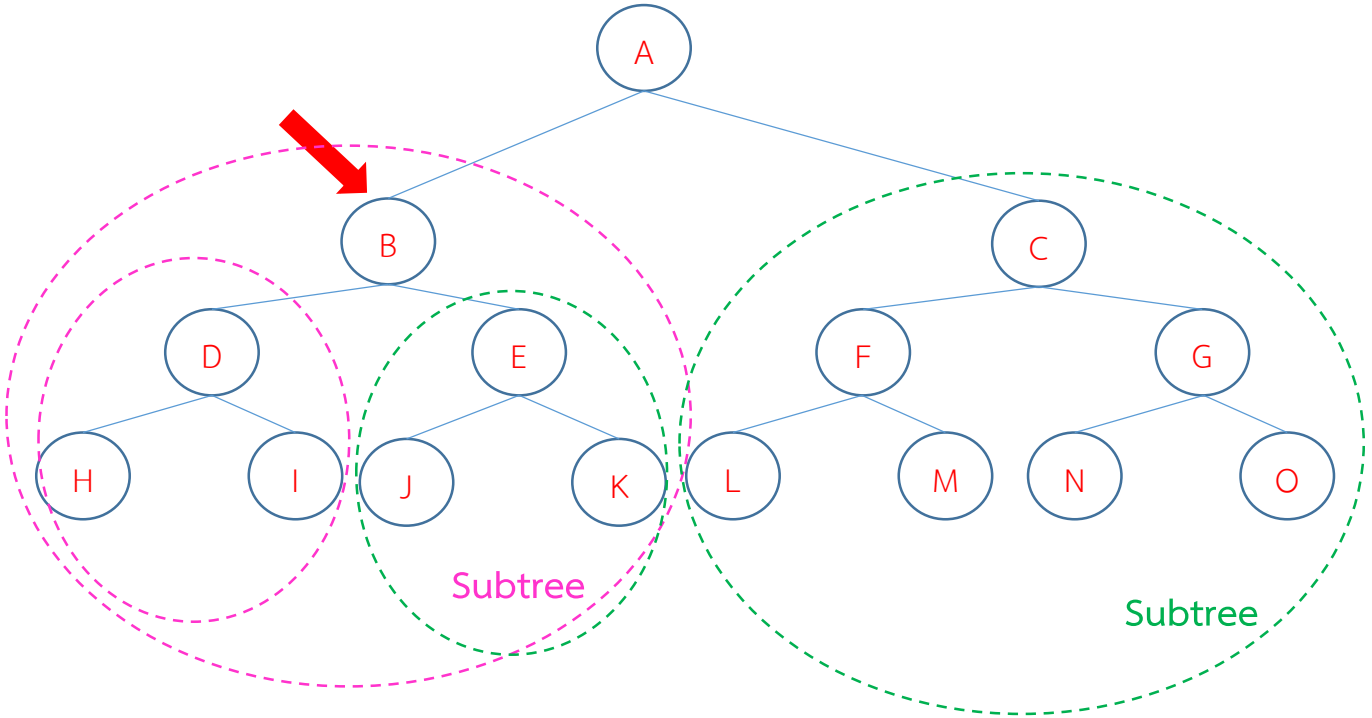
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

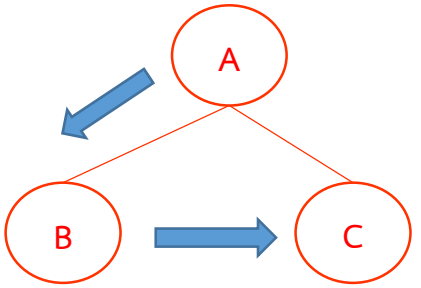
A --> B --> C



A,B

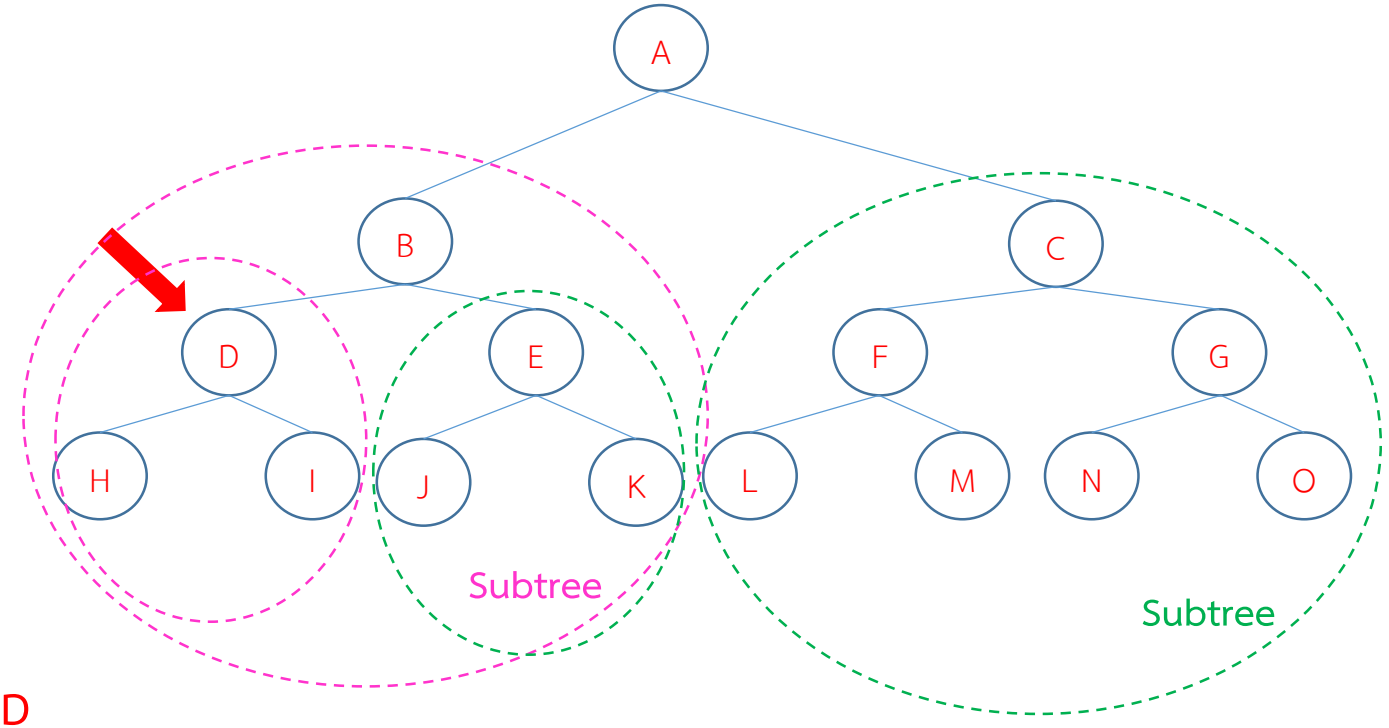
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

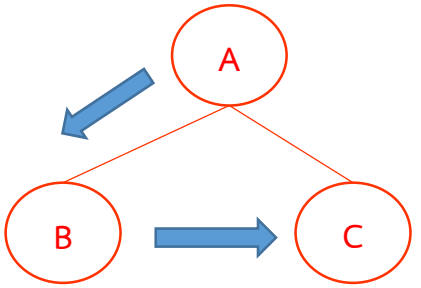
A --> B --> C



A,B,D

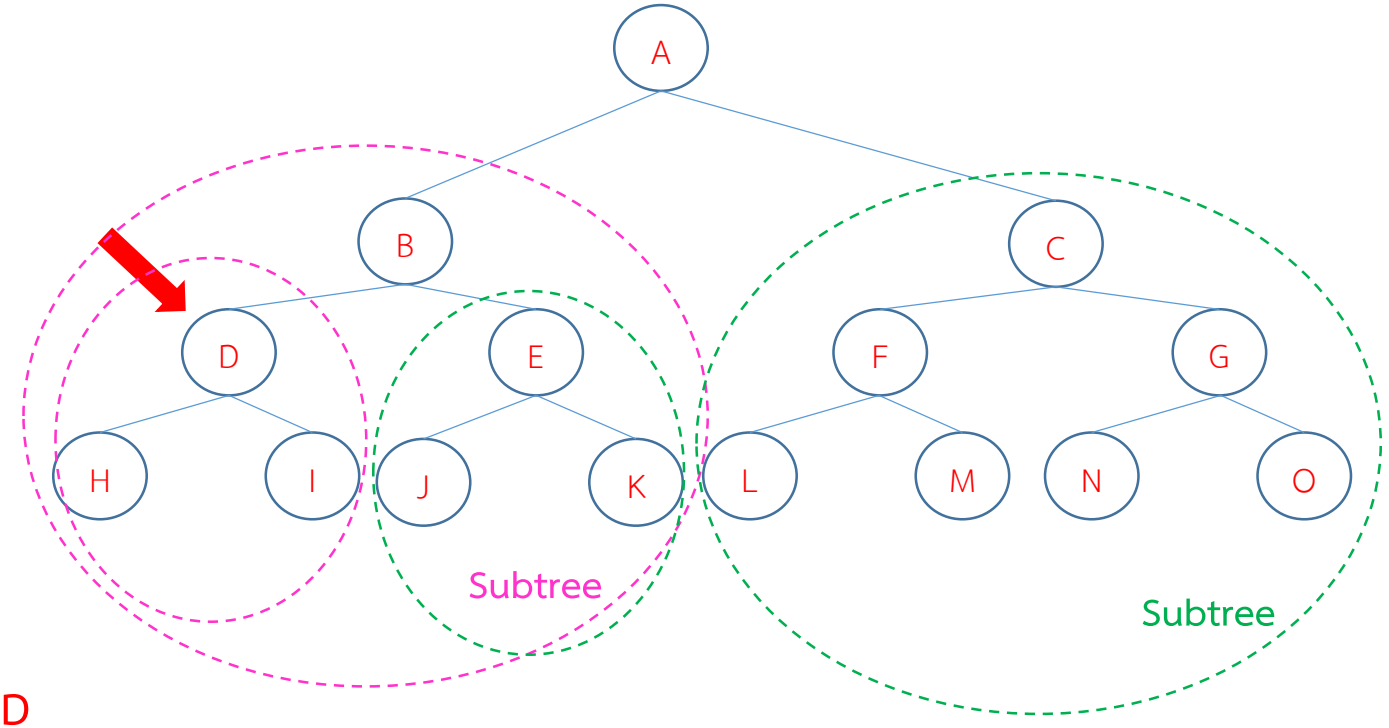
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

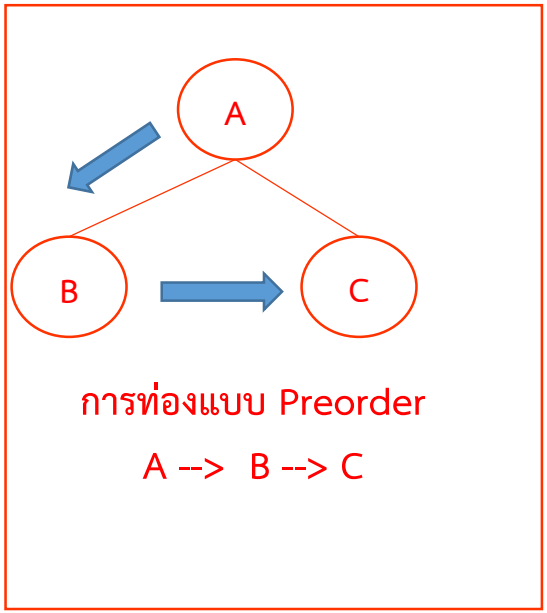
A --> B --> C

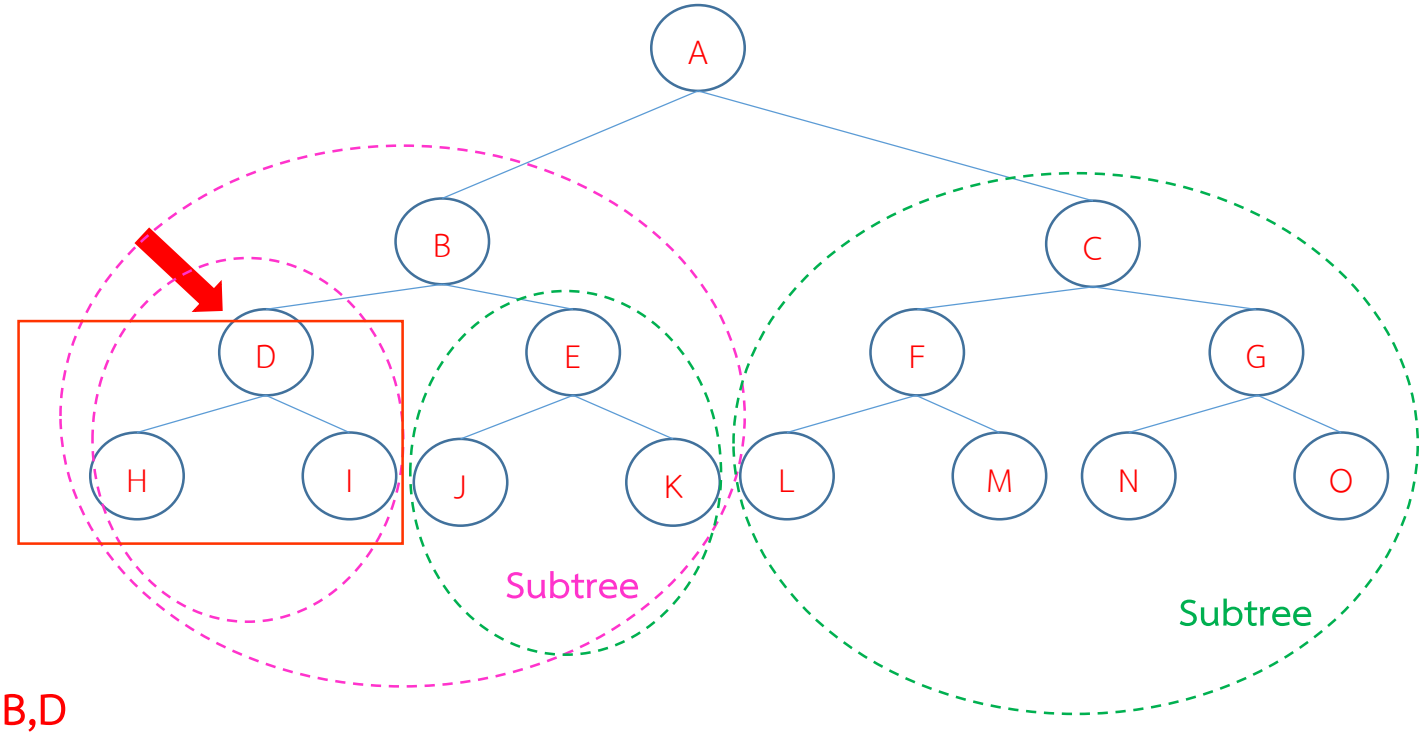


A,B,D

Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder

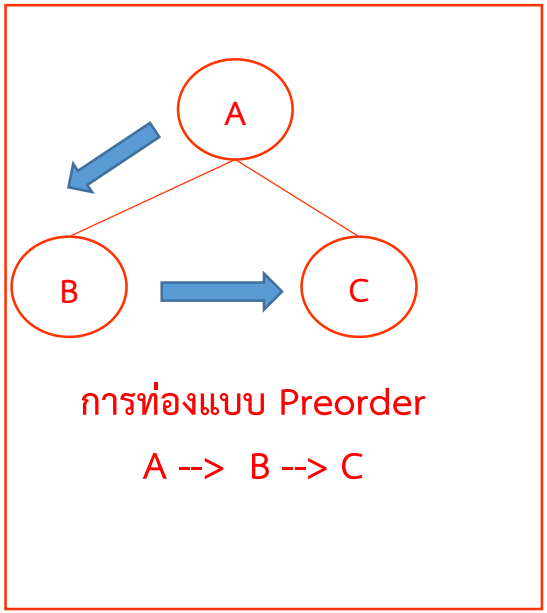


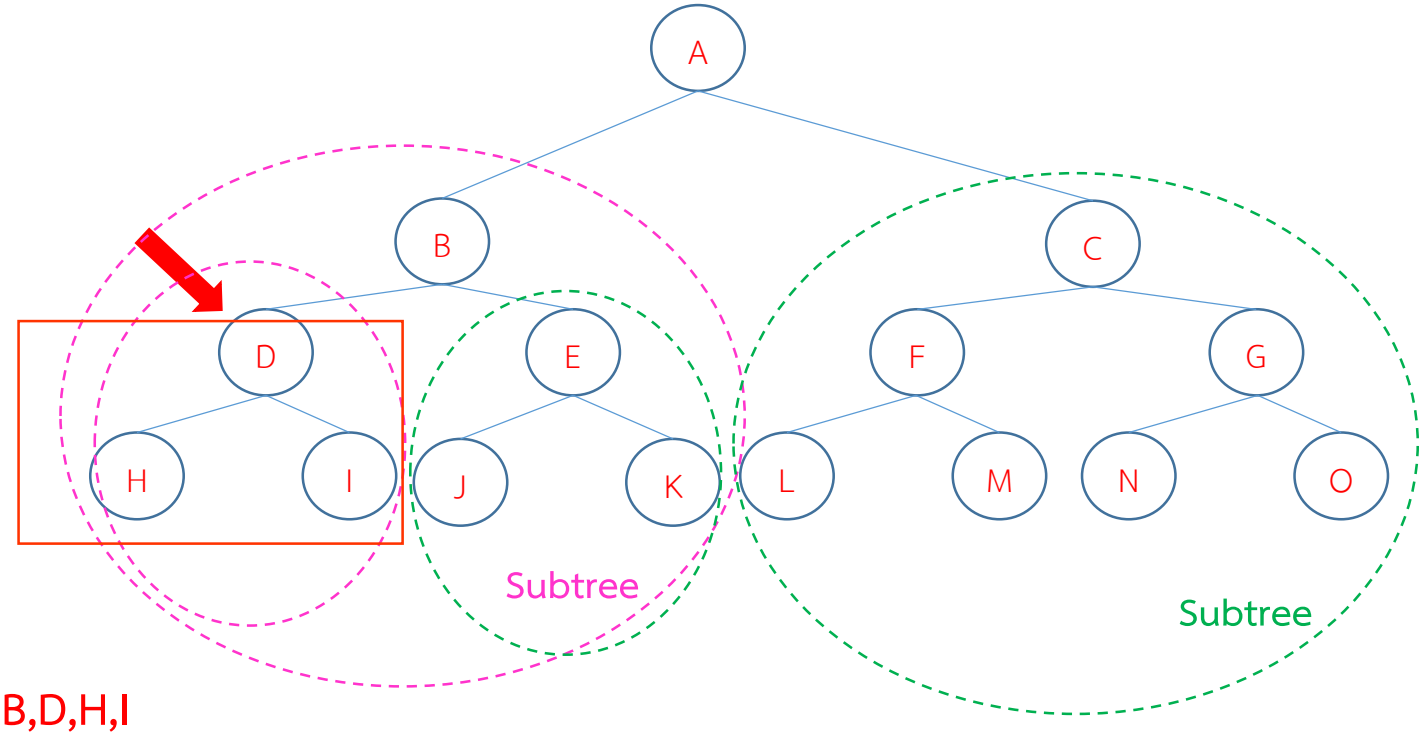


A,B,D

Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder

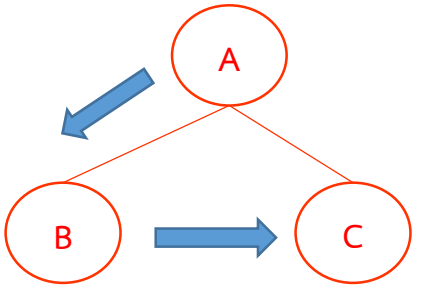




A,B,D,H,I

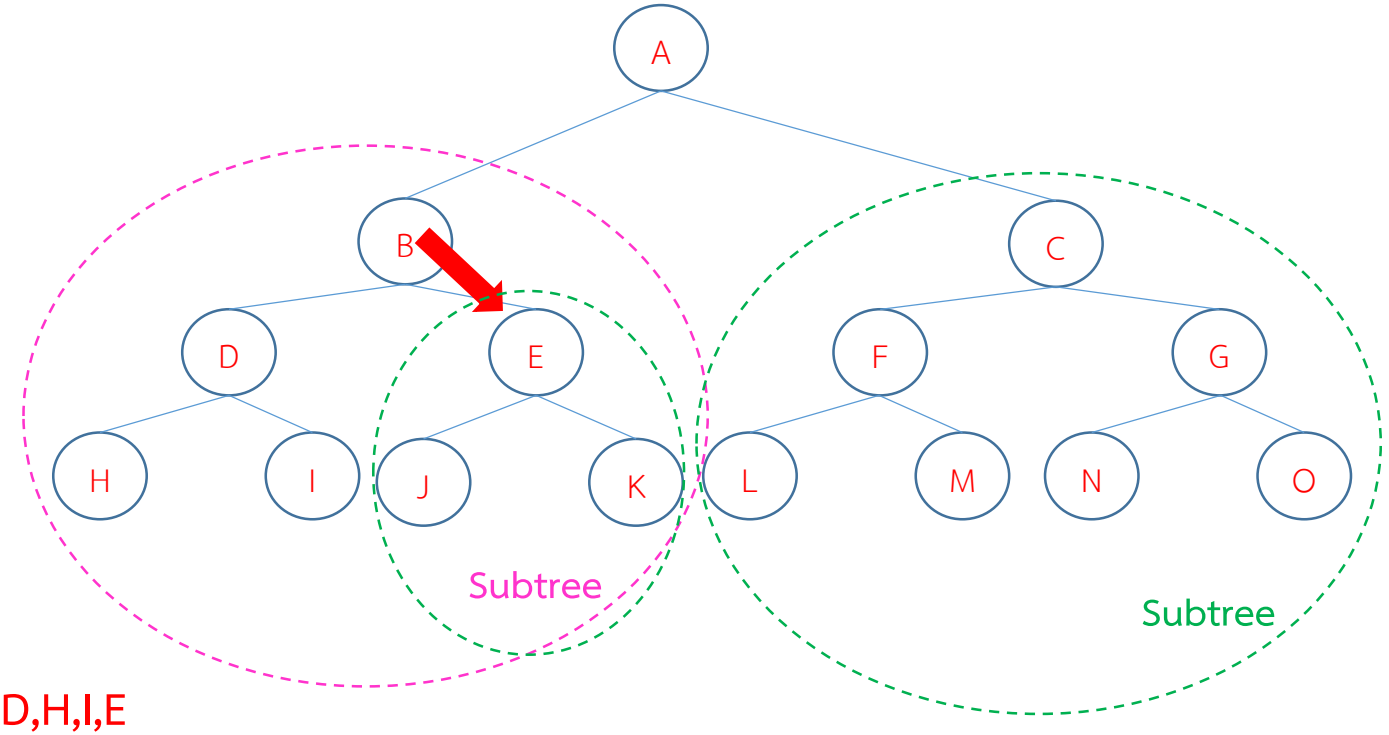
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

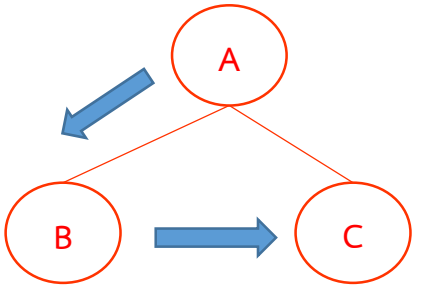
A --> B --> C



A,B,D,H,I,E

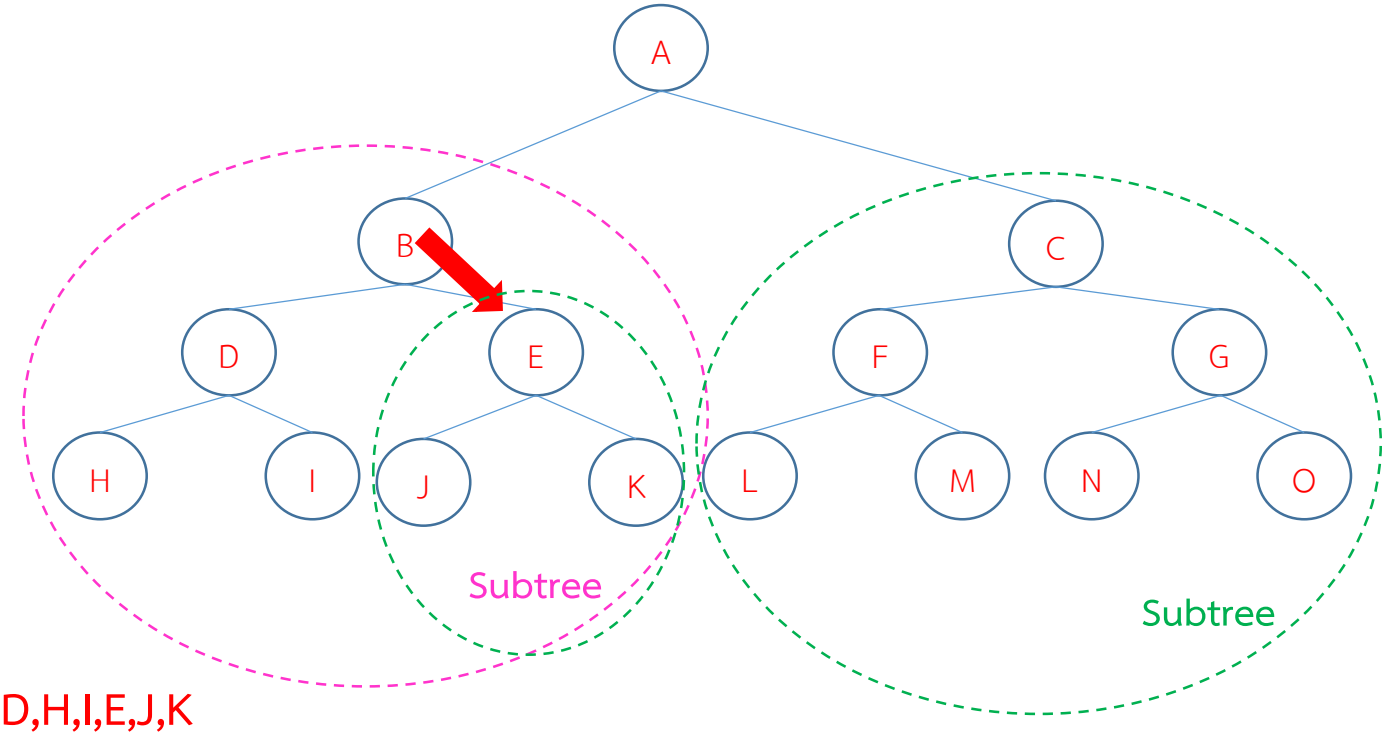
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

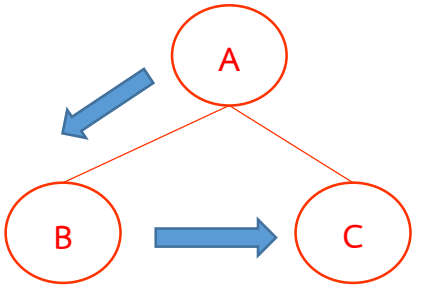
A --> B --> C



A,B,D,H,I,E,J,K

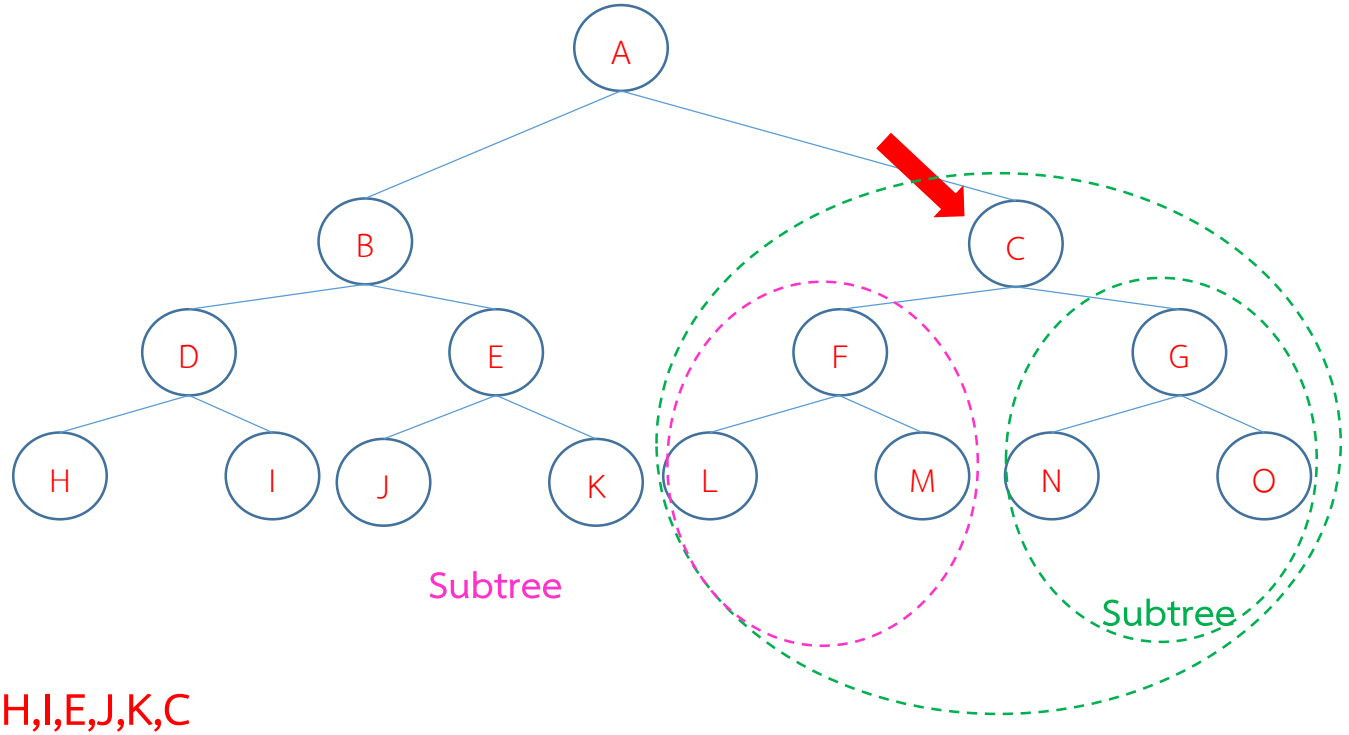
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

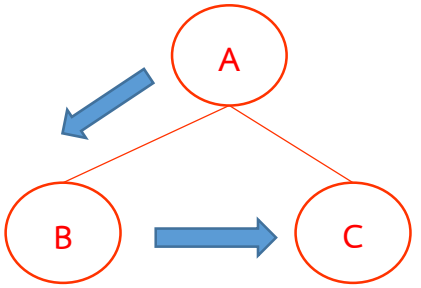
A --> B --> C



A,B,D,H,I,E,J,K,C

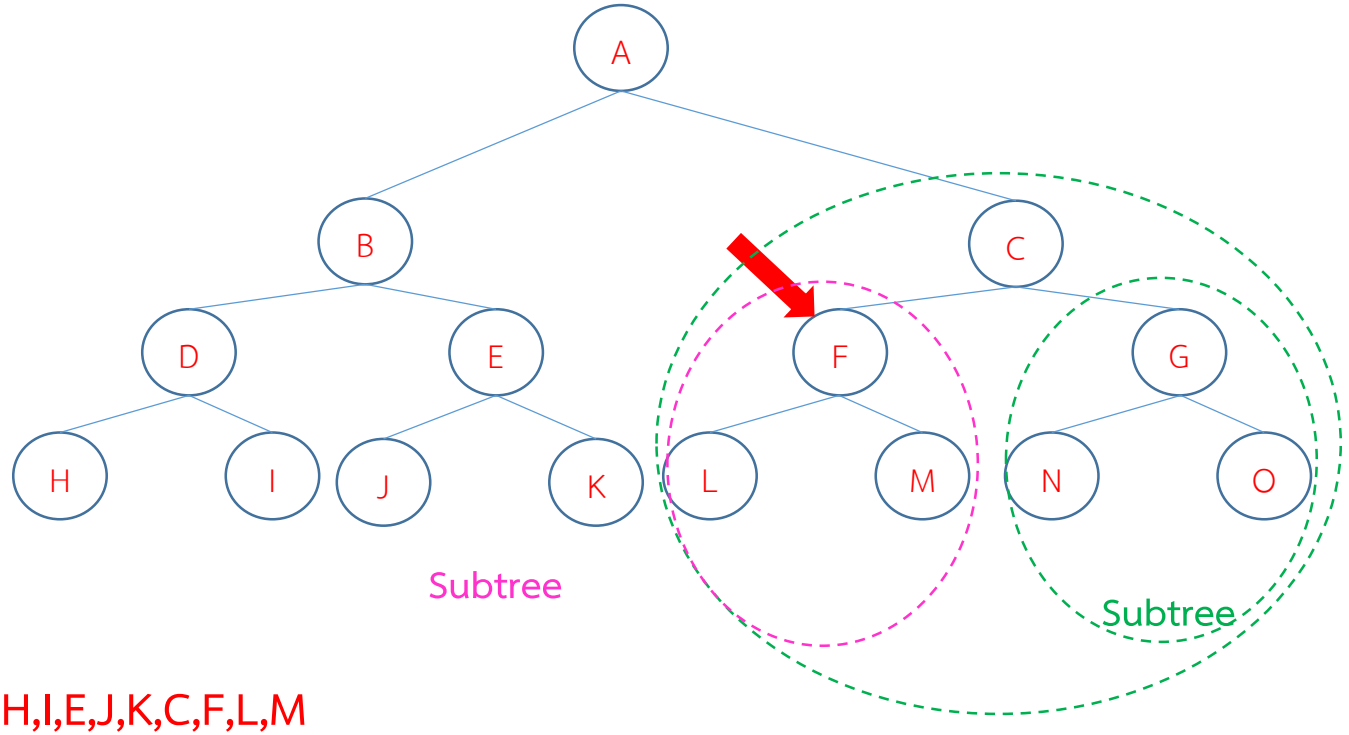
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

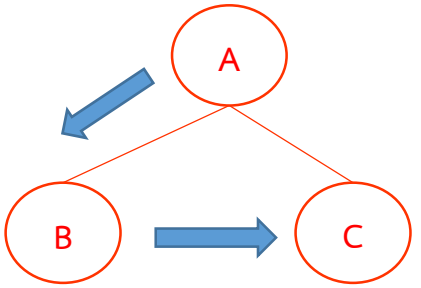
A --> B --> C



A,B,D,H,I,E,J,K,C,F,L,M

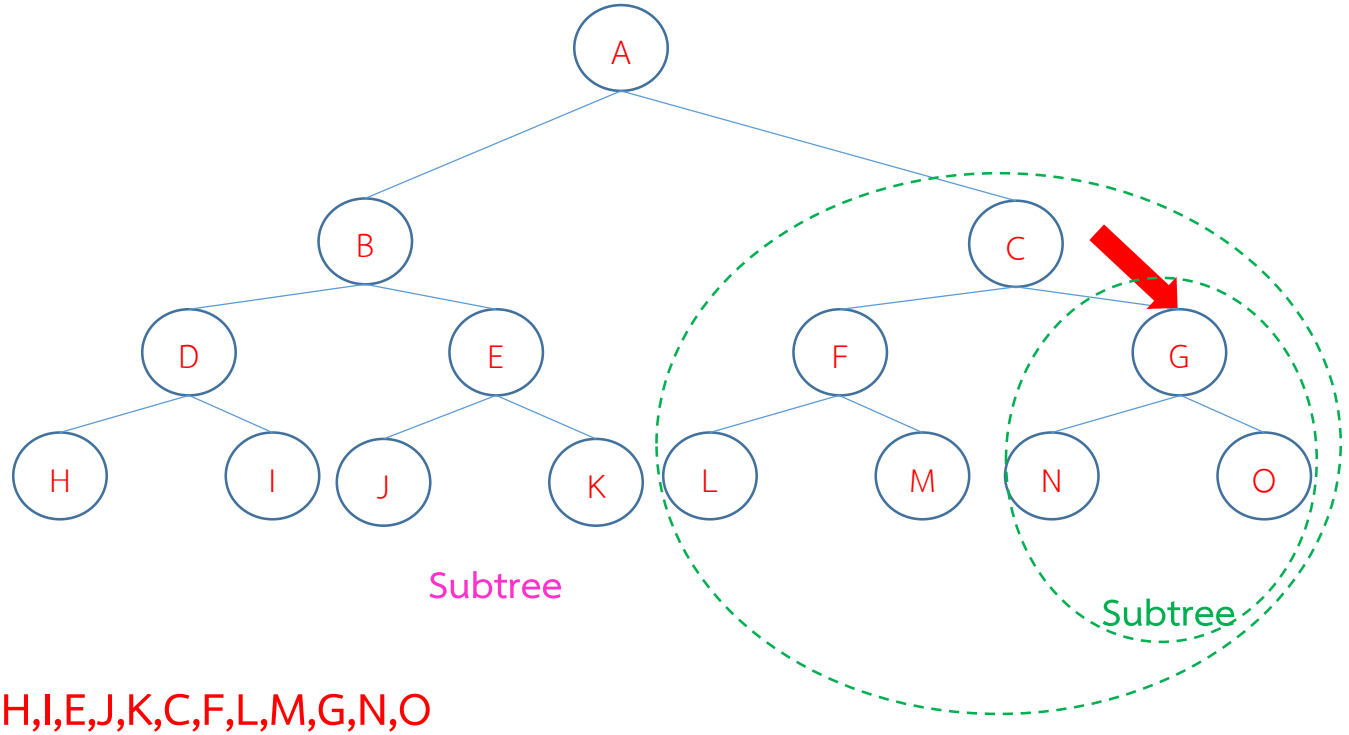
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

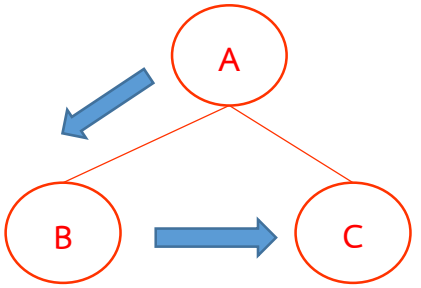
A --> B --> C



A,B,D,H,I,E,J,K,C,F,L,M,G,N,O

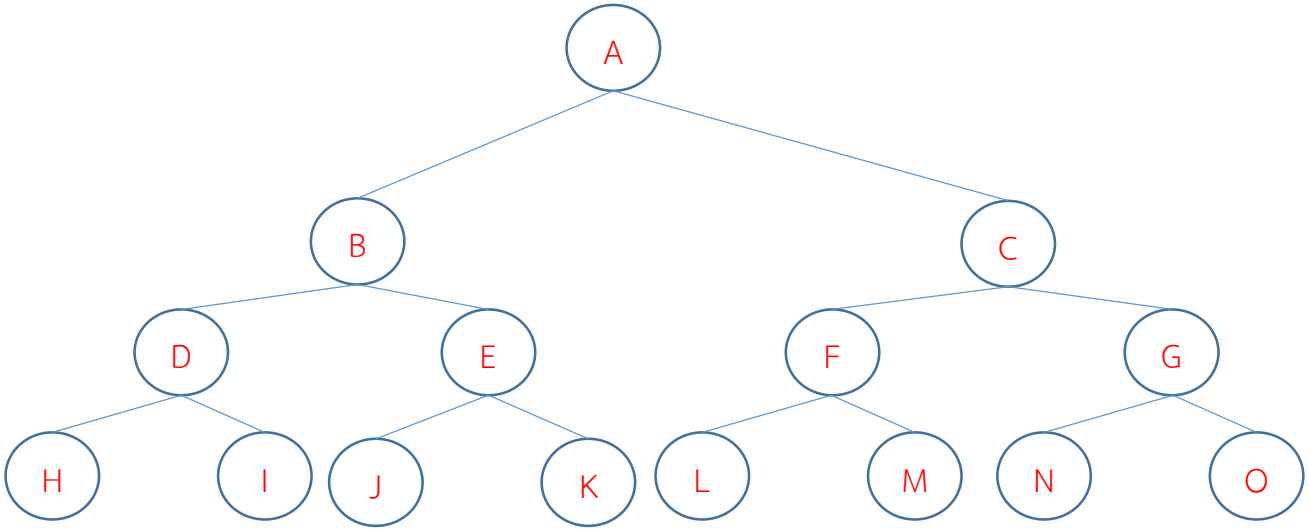
Preorder Traversal

1) เริ่มจาก Root Node
2) ไต่ไปทาง subtree ด้านซ้ายแบบ Preorder
3) ไต่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

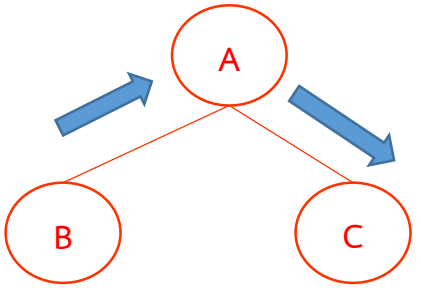
A --> B --> C



A,B,D,H,I,E,J,K,C,F,L,M,G,N,O

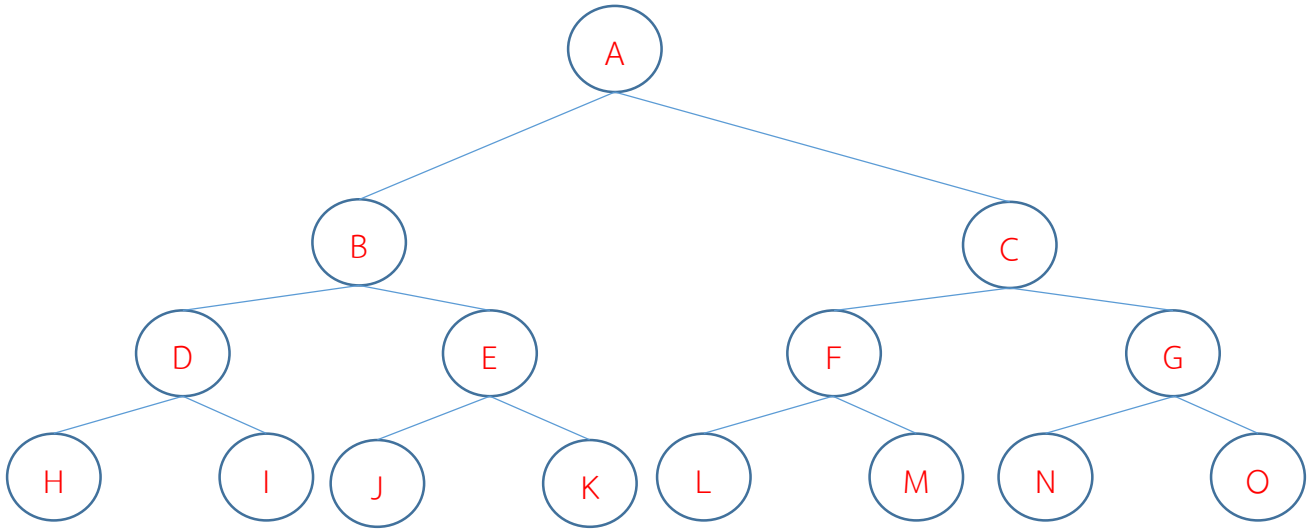
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



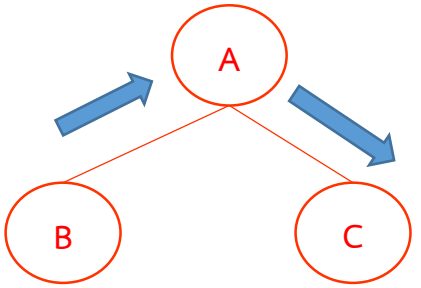
การท่องแบบ Inorder

B --> A --> C



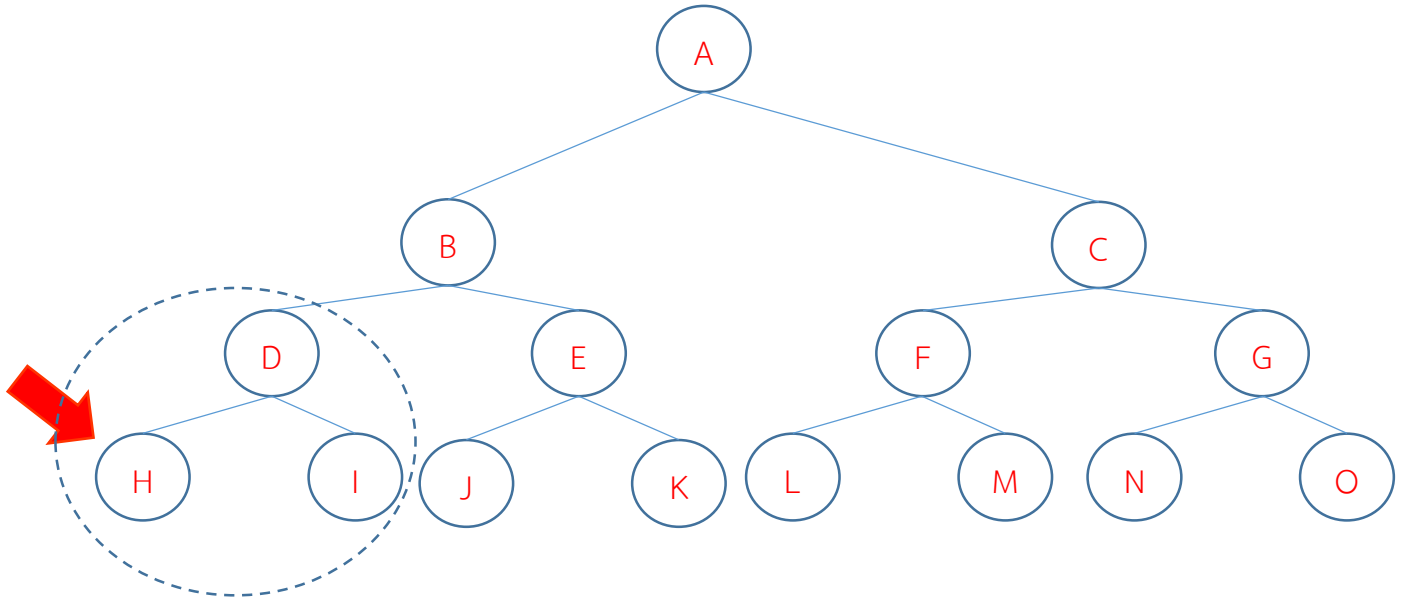
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

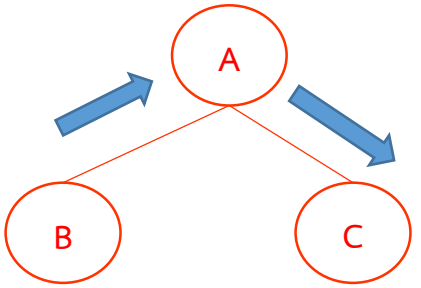
B --> A --> C



H

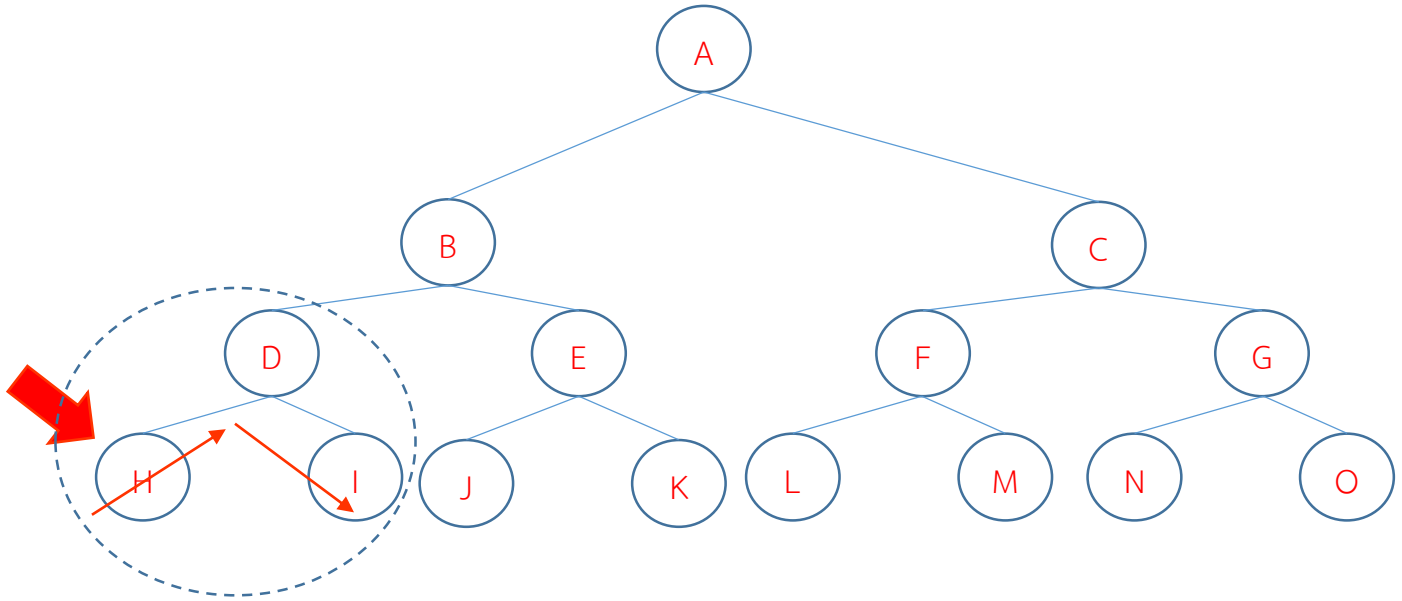
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

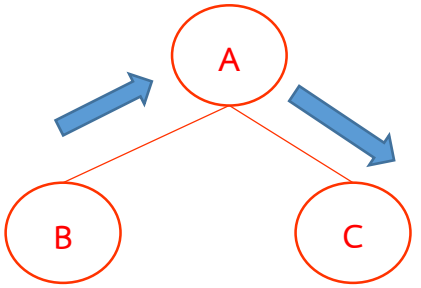
B --> A --> C



H

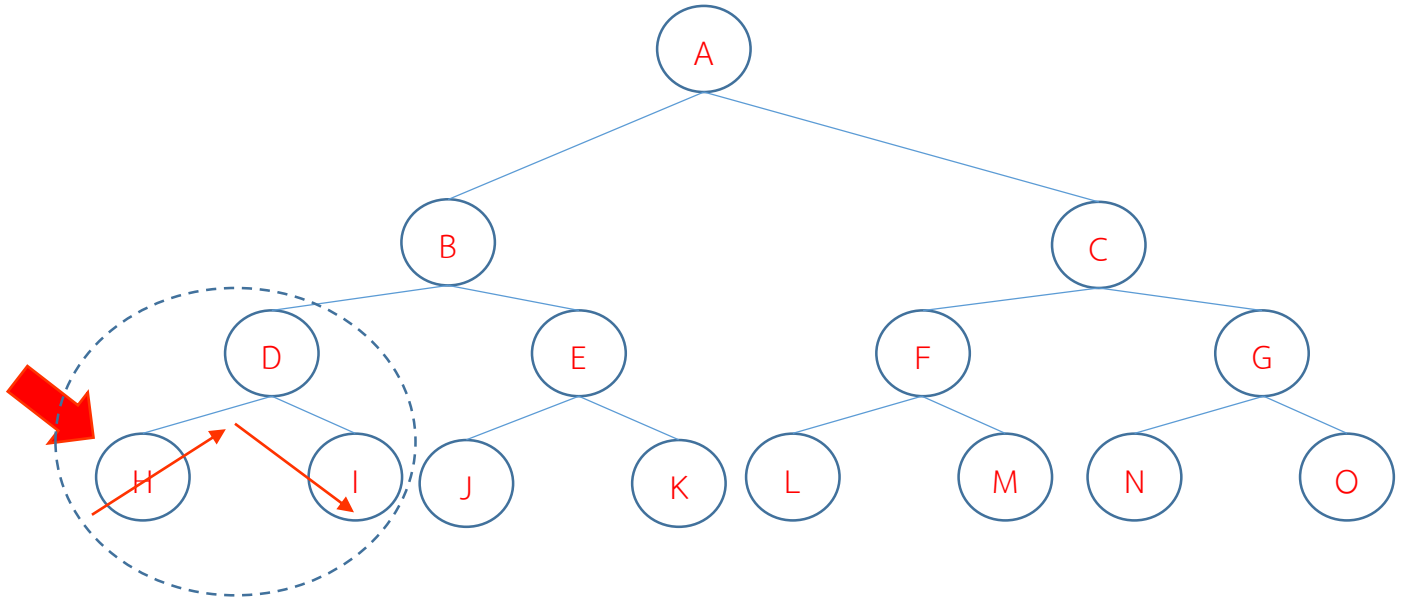
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

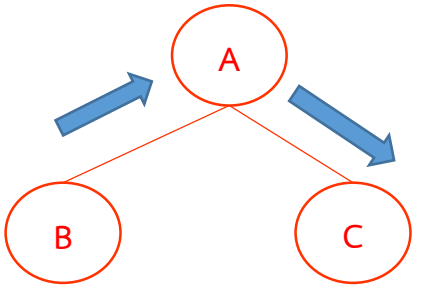
B --> A --> C



H,D,I

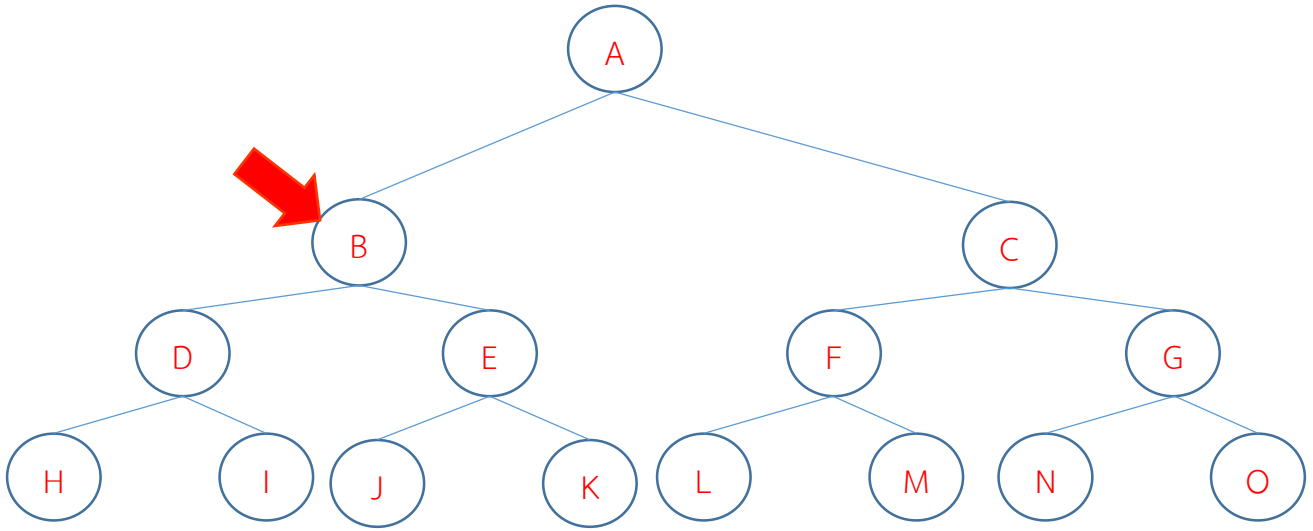
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

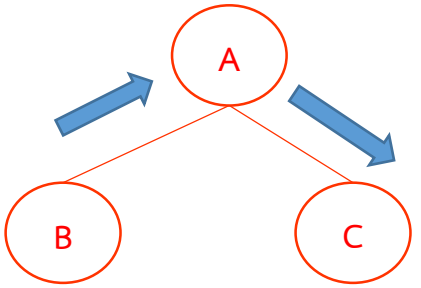
B --> A --> C



H,D,I,B

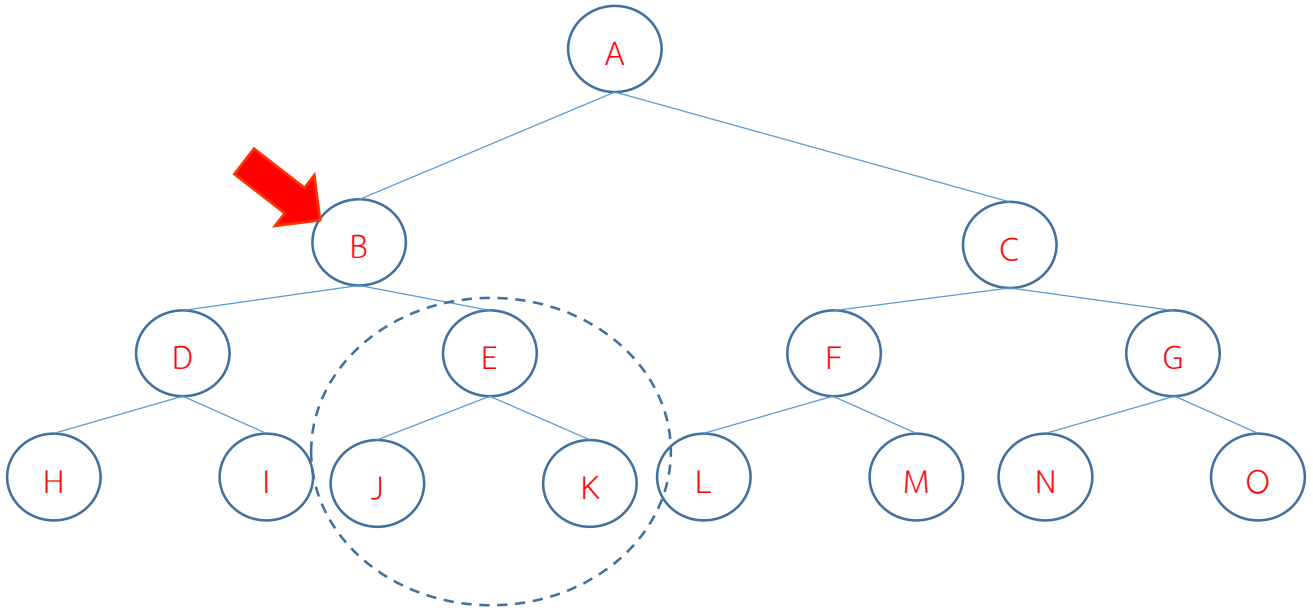
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

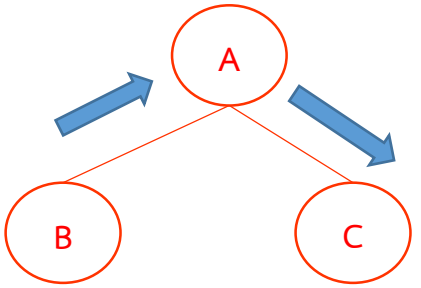
B --> A --> C



H,D,I,B

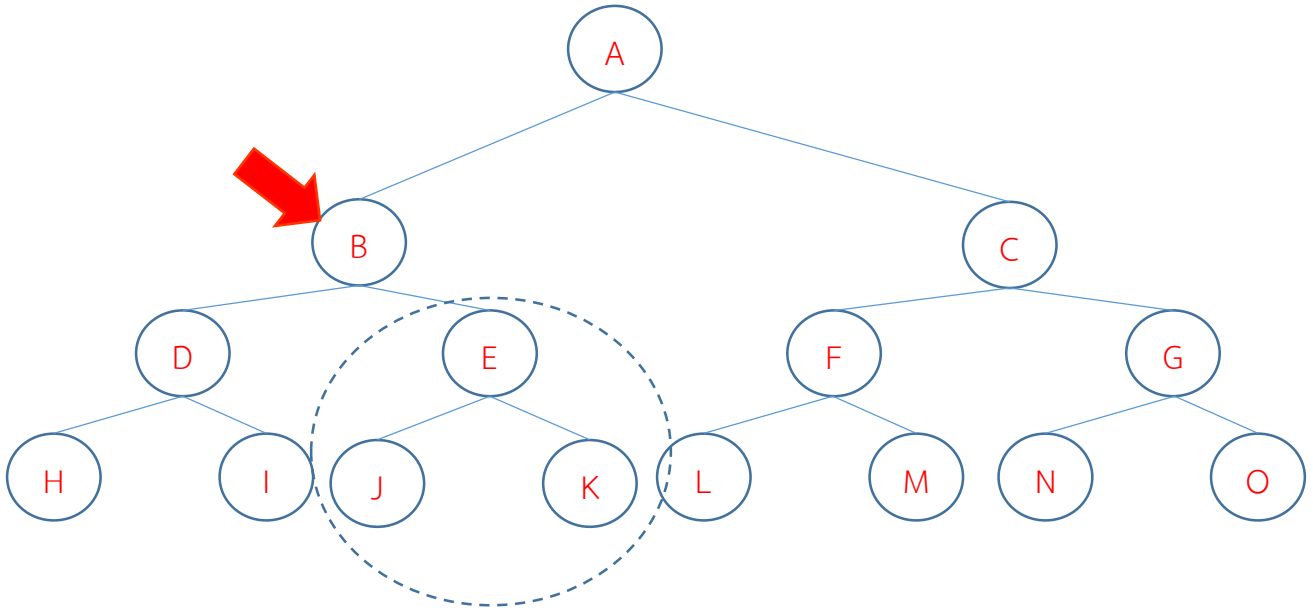
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

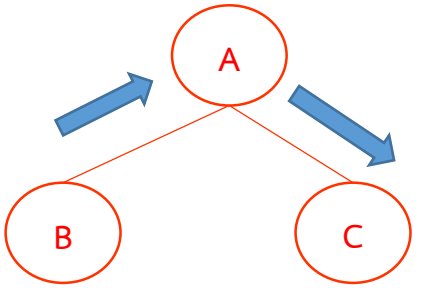
B --> A --> C



H,D,I,B,J,E,K

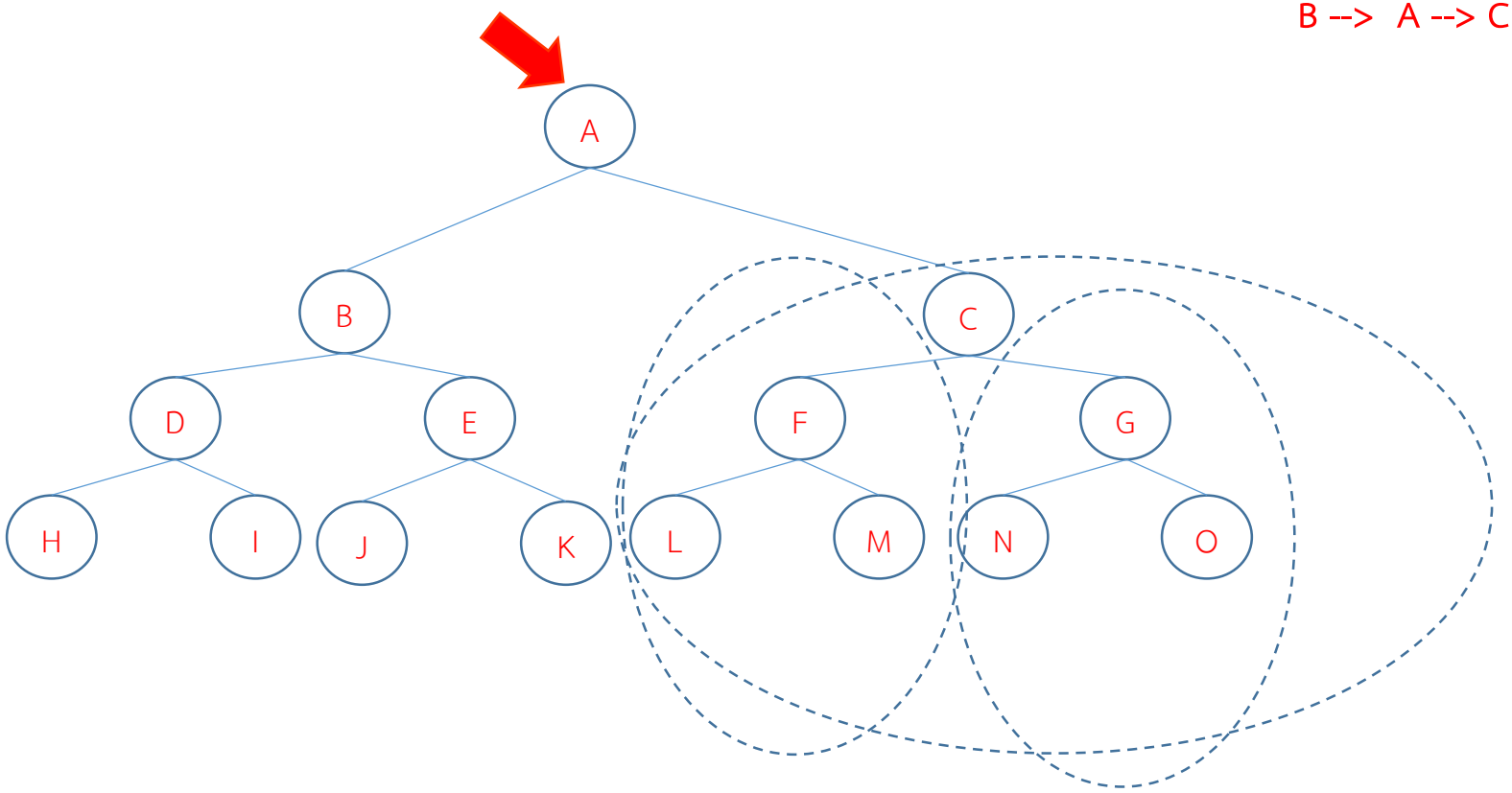
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

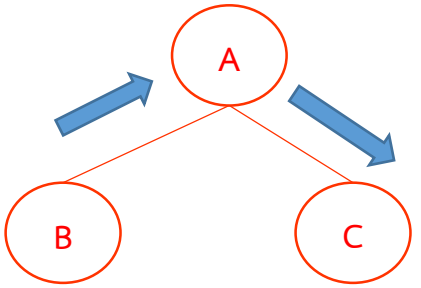
B --> A --> C



H,D,I,B,J,E,K,A

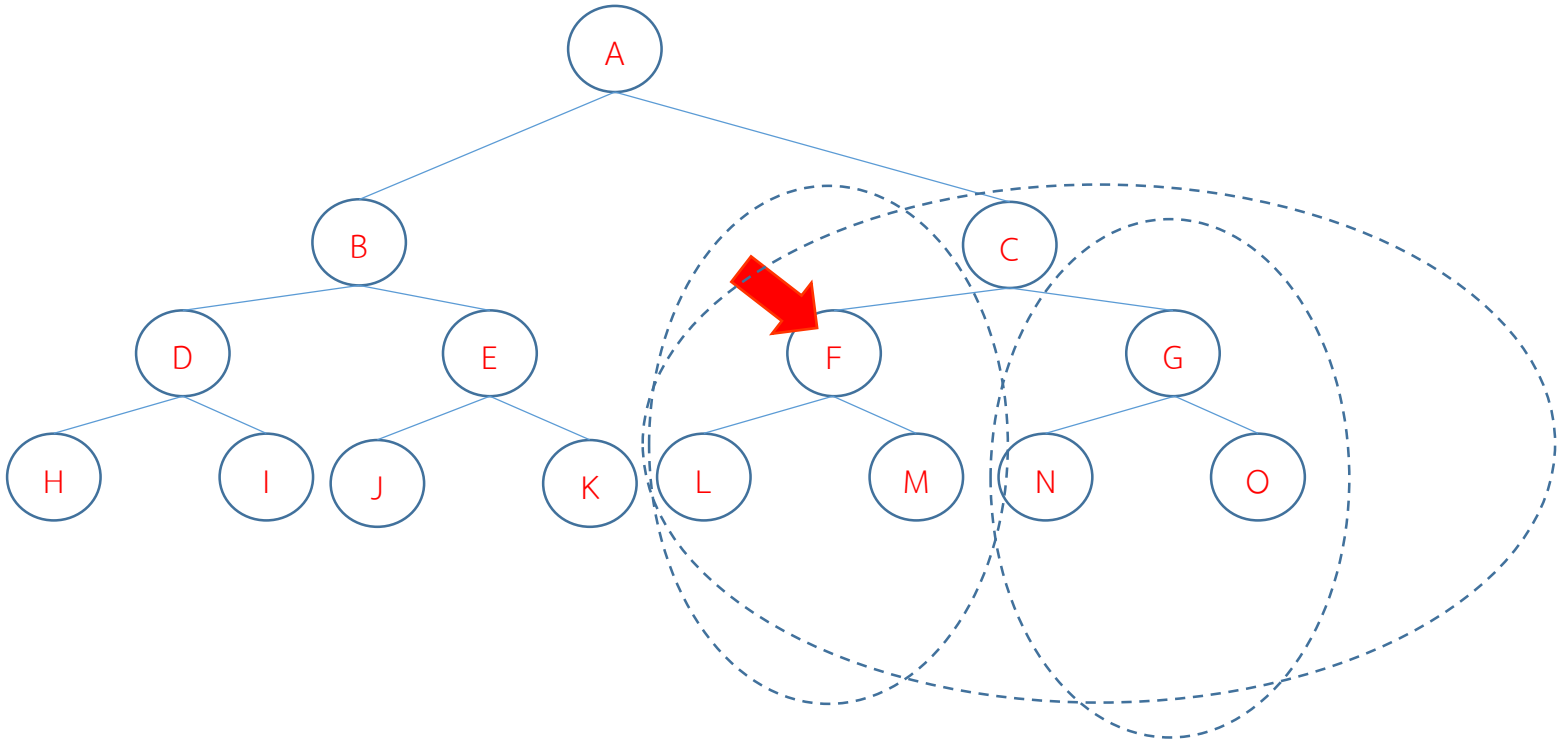
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

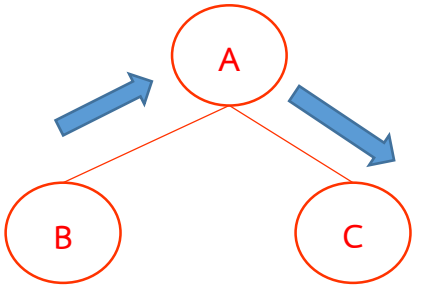
B --> A --> C



H,D,I,B,J,E,K,A

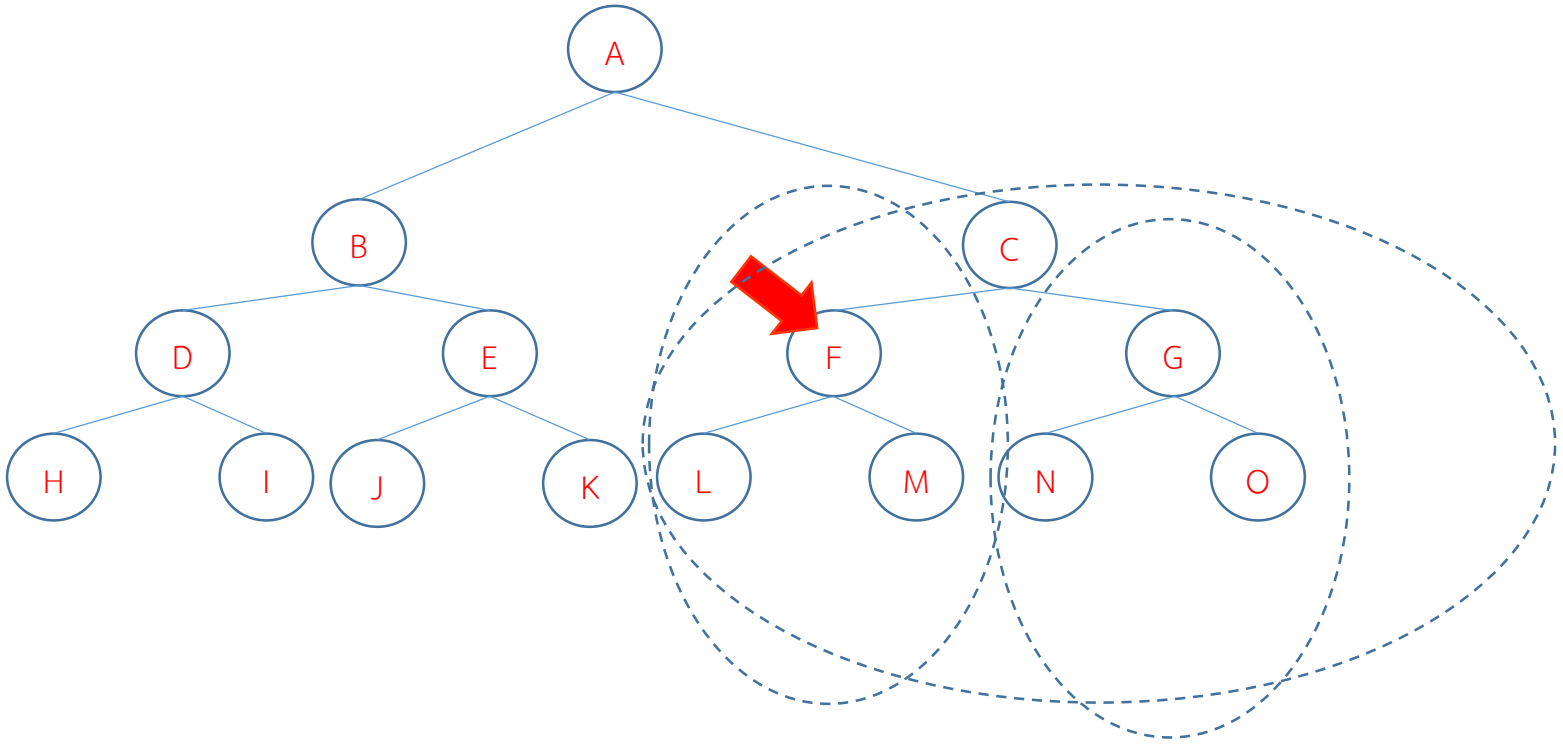
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

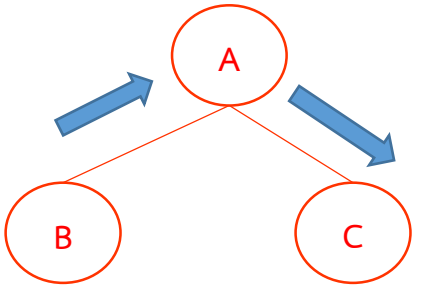
B --> A --> C



H,D,I,B,J,E,K,A,L,F,M

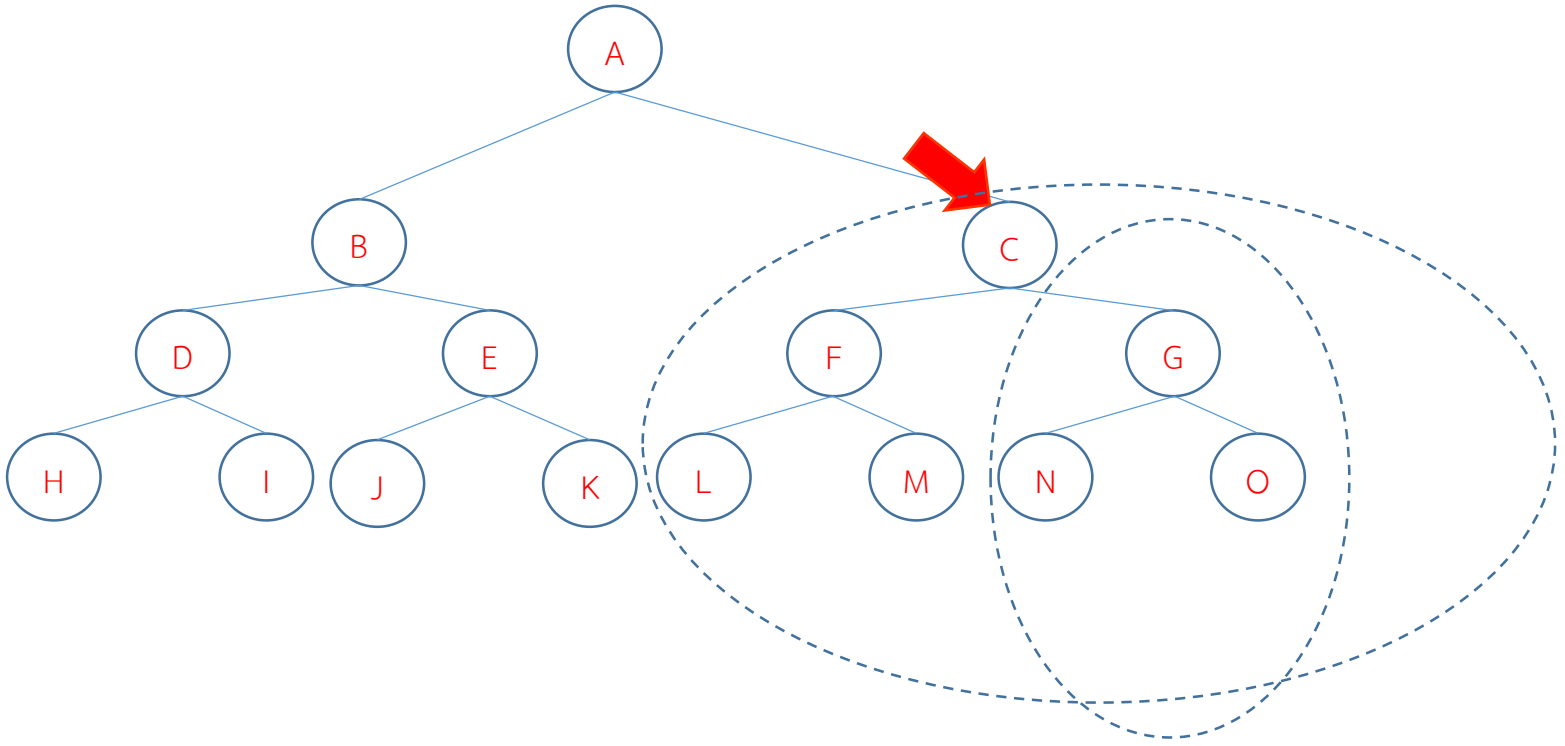
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

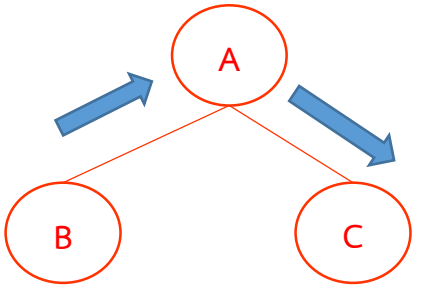
B --> A --> C



H,D,I,B,J,E,K,A,L,F,M,C

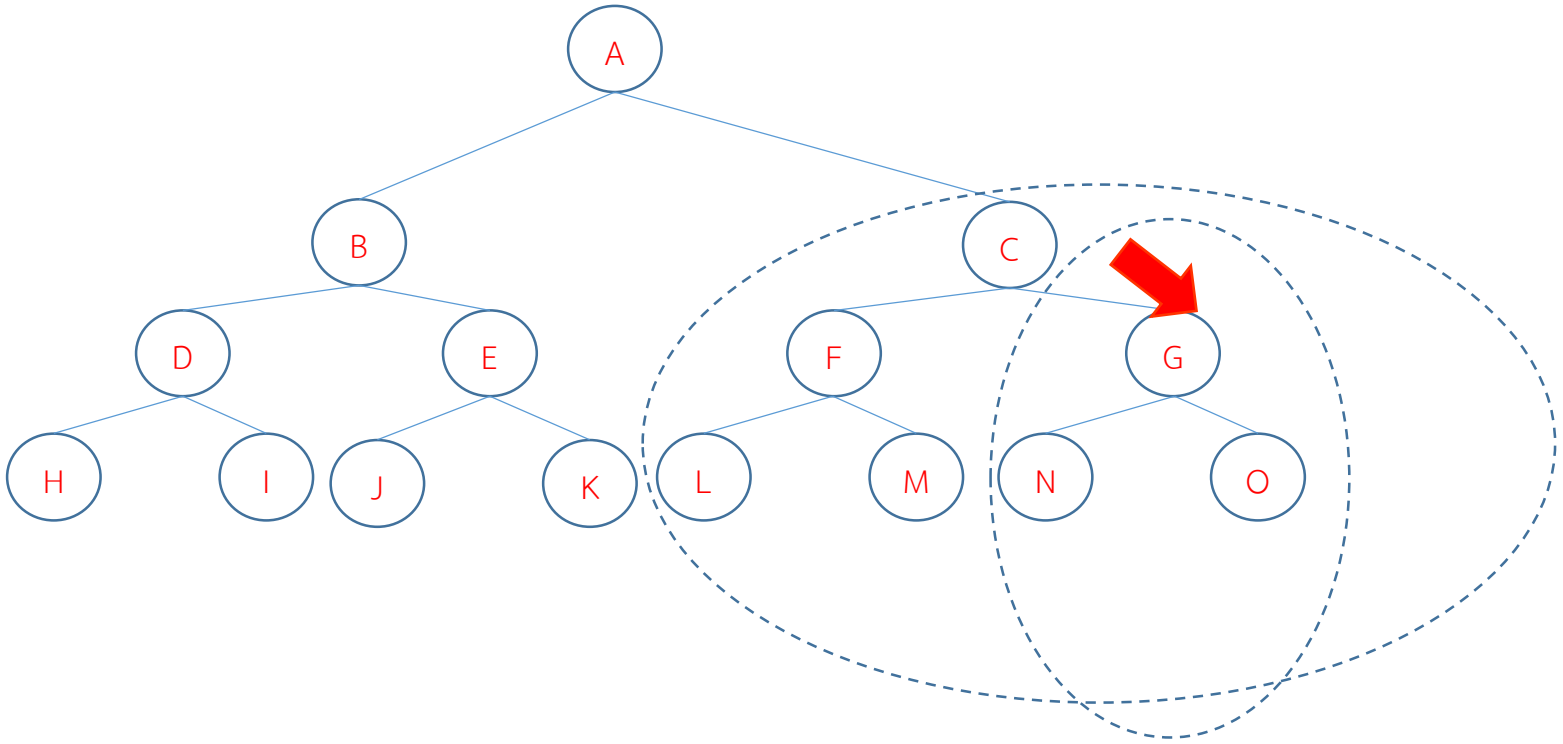
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

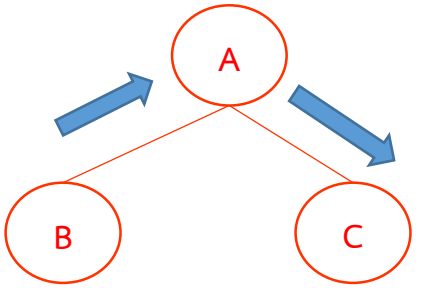
B --> A --> C



H,D,I,B,J,E,K,A,L,F,M,C

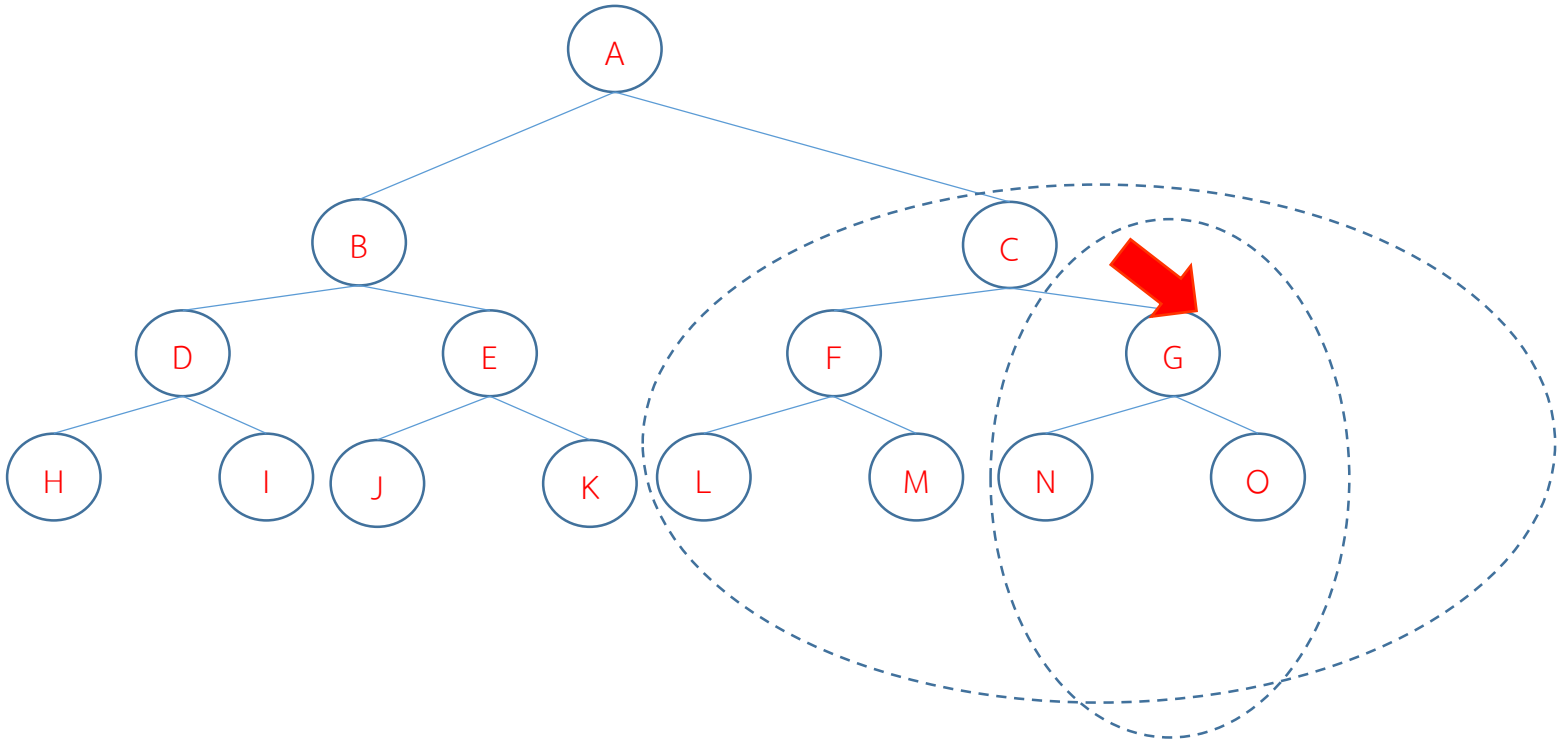
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

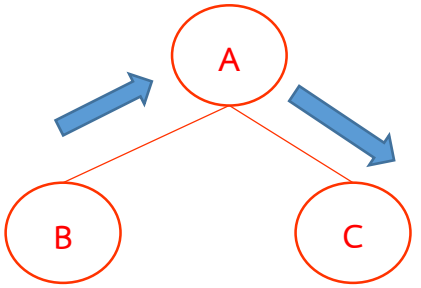
B --> A --> C



H,D,I,B,J,E,K,A,L,F,M,C,N,G,O

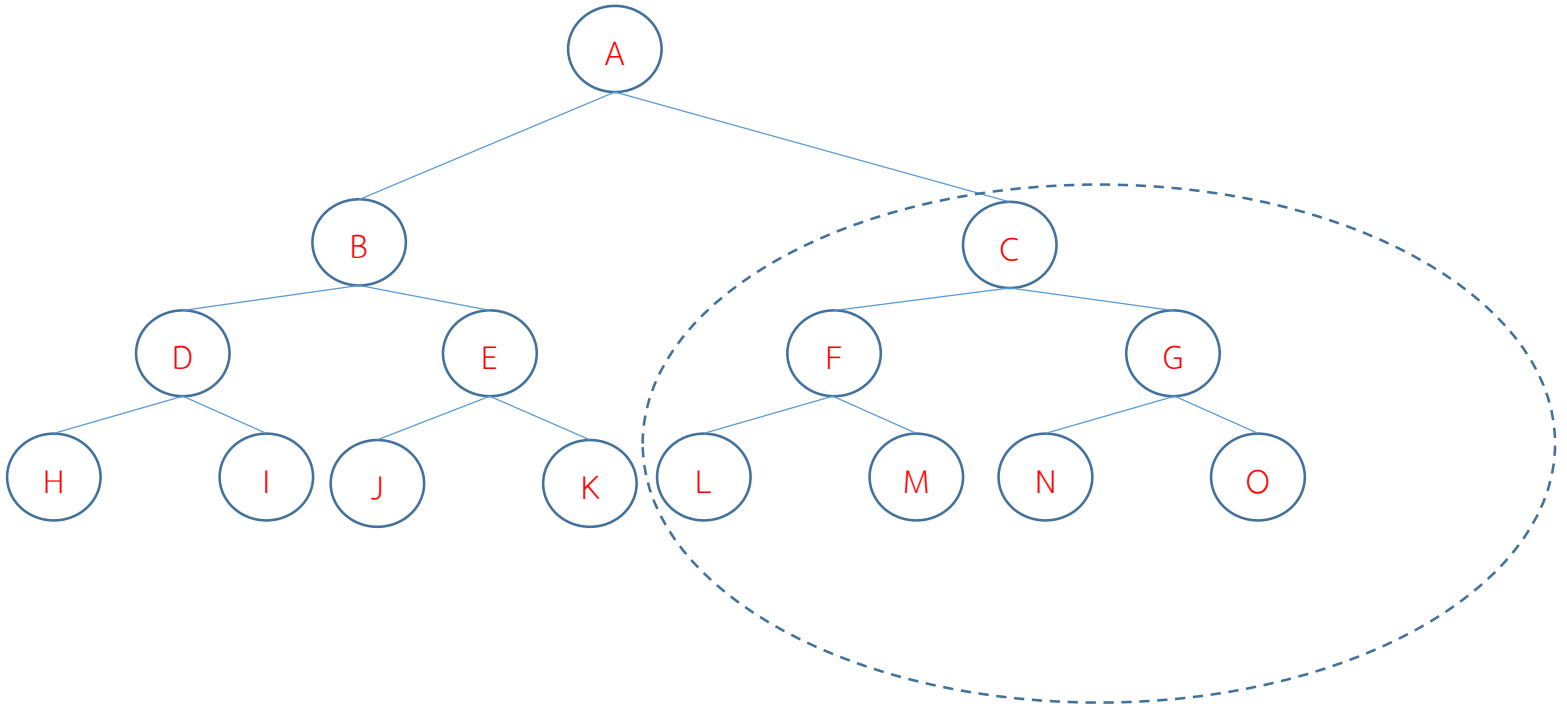
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

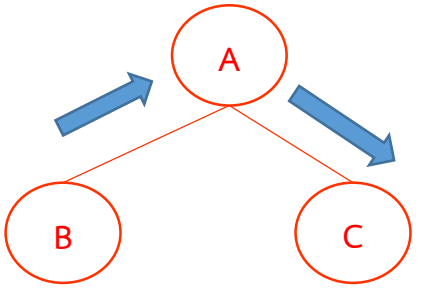
B --> A --> C



H,D,I,B,J,E,K,A,L,F,M,C,N,G,O

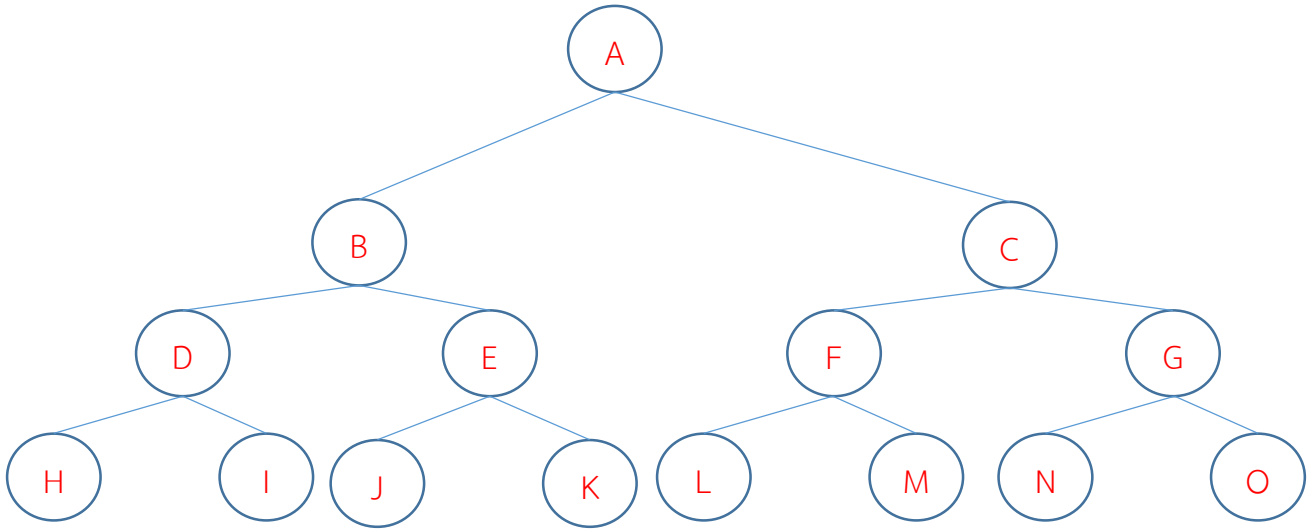
Inorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ย้อนขึ้นไปยัง Root ของ Subtree
3) ไต่ไปทาง subtree ด้านขวาแบบ Inorder



การท่องแบบ Inorder

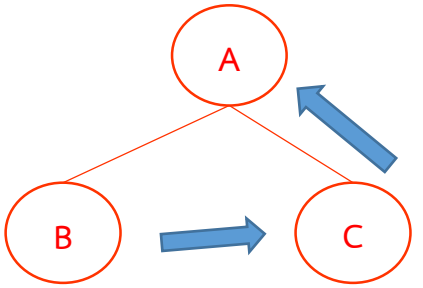
B --> A --> C



H,D,I,B,J,E,K,A,L,F,M,C,N,G,O

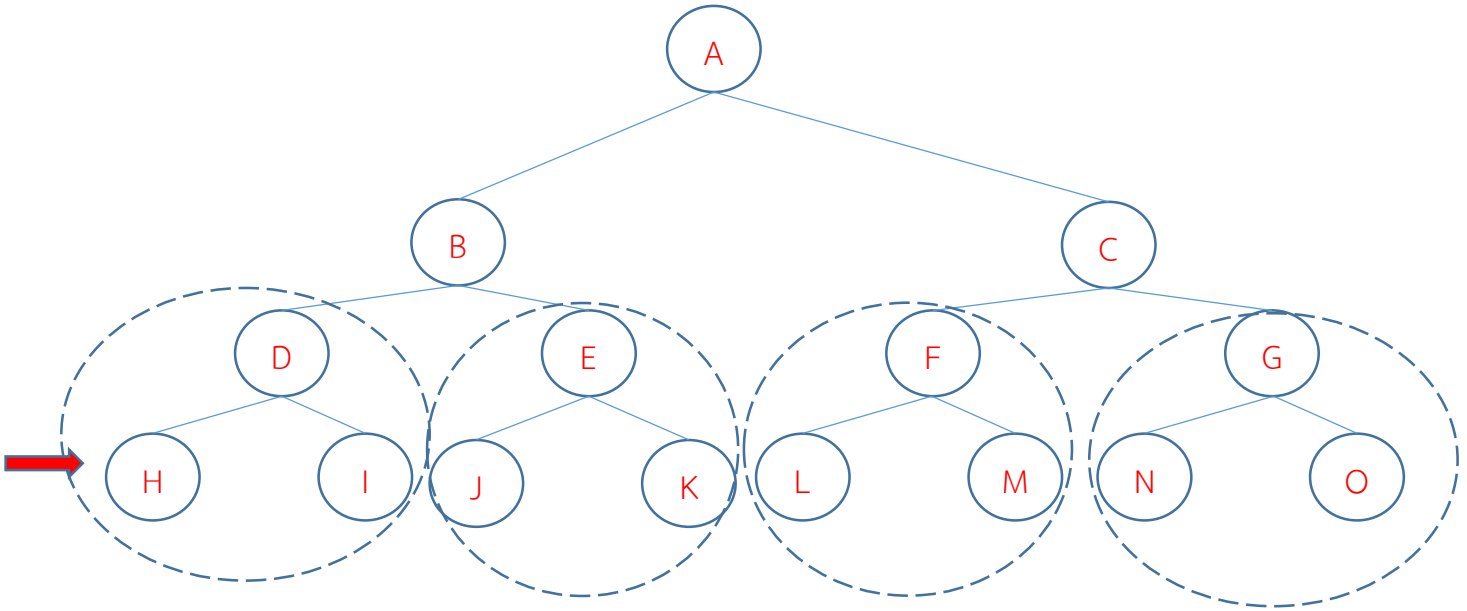
Postorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ไต่ไปทาง subtree ด้านขวาแบบ Postorder
3) ย้อนขึ้นไปยัง Root นกำ



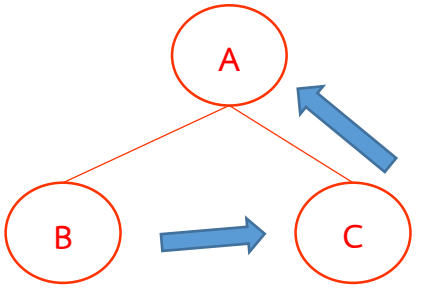
การท่องแบบ Inorder

B --> C --> A



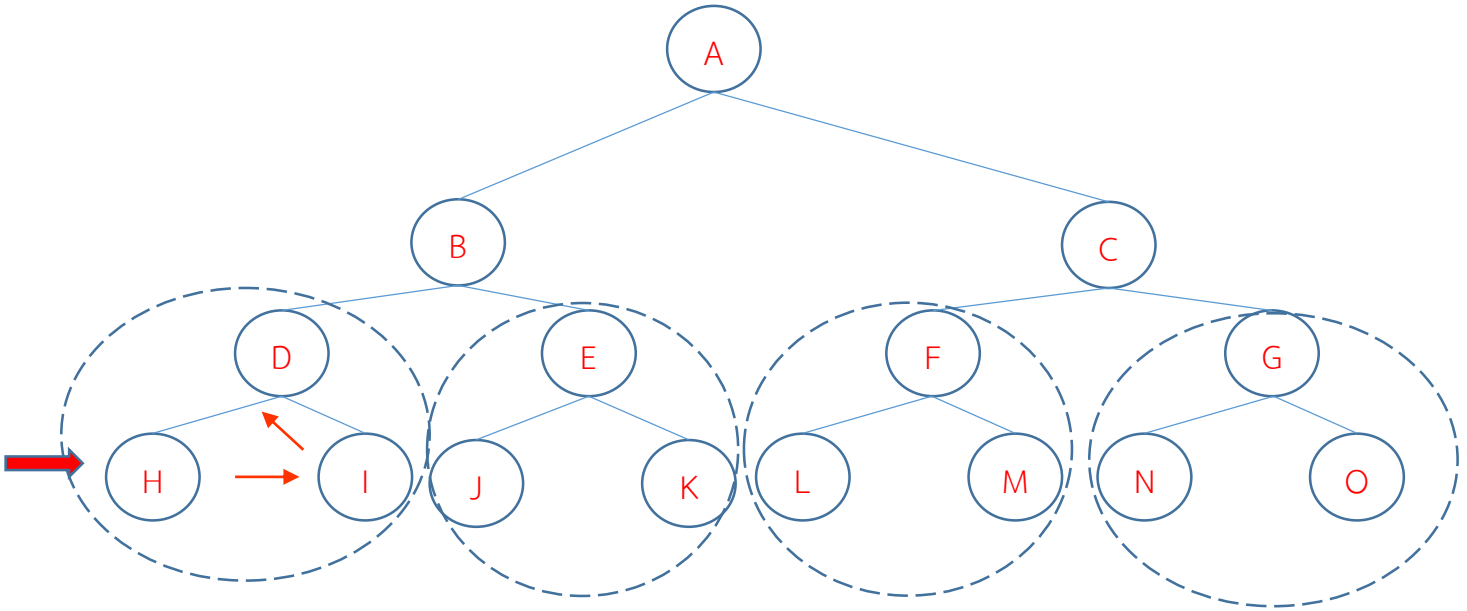
Postorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ไต่ไปทาง subtree ด้านขวาแบบ Postorder
3) ย้อนขึ้นไปยัง Root นกำ



การท่องแบบ Inorder

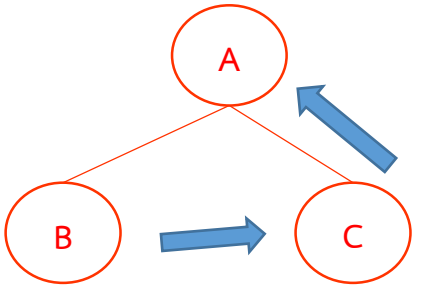
B --> C --> A



H,I,D

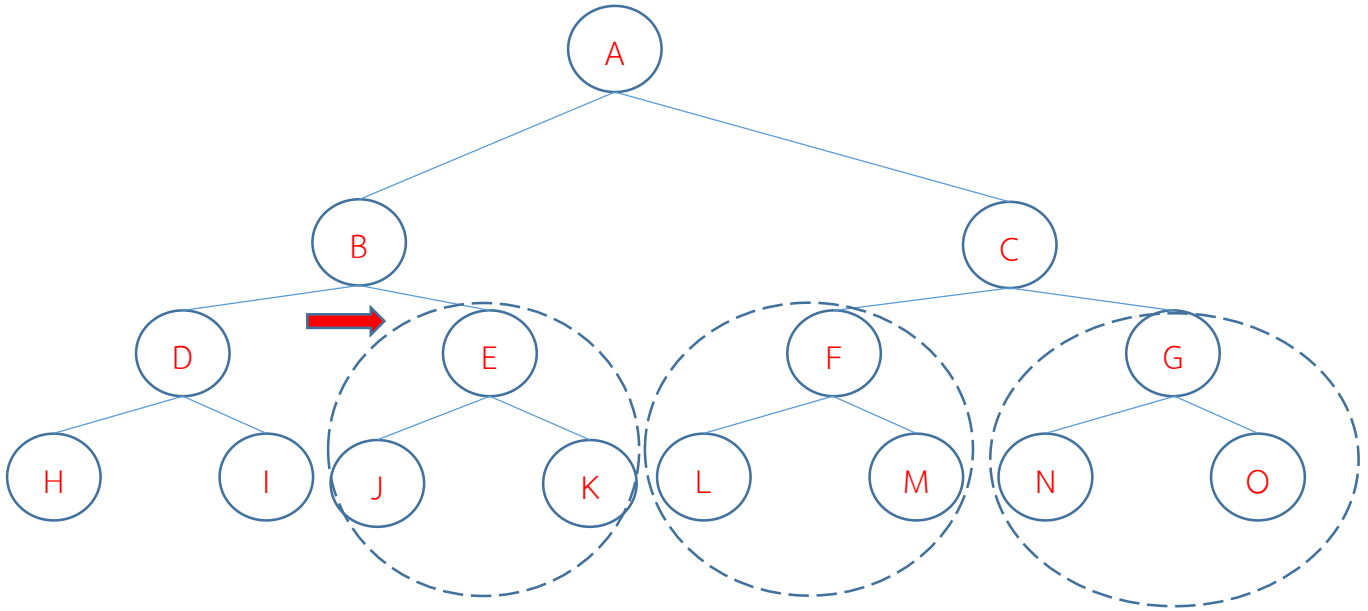
Postorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ไต่ไปทาง subtree ด้านขวาแบบ Postorder
3) ย้อนขึ้นไปยัง Root นกำ



การท่องแบบ Inorder

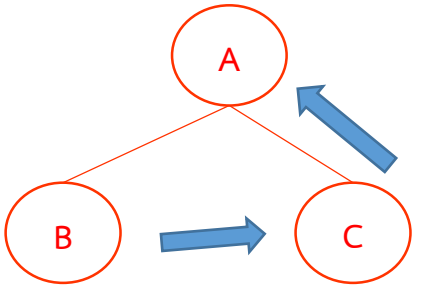
B --> C --> A



H,I,D

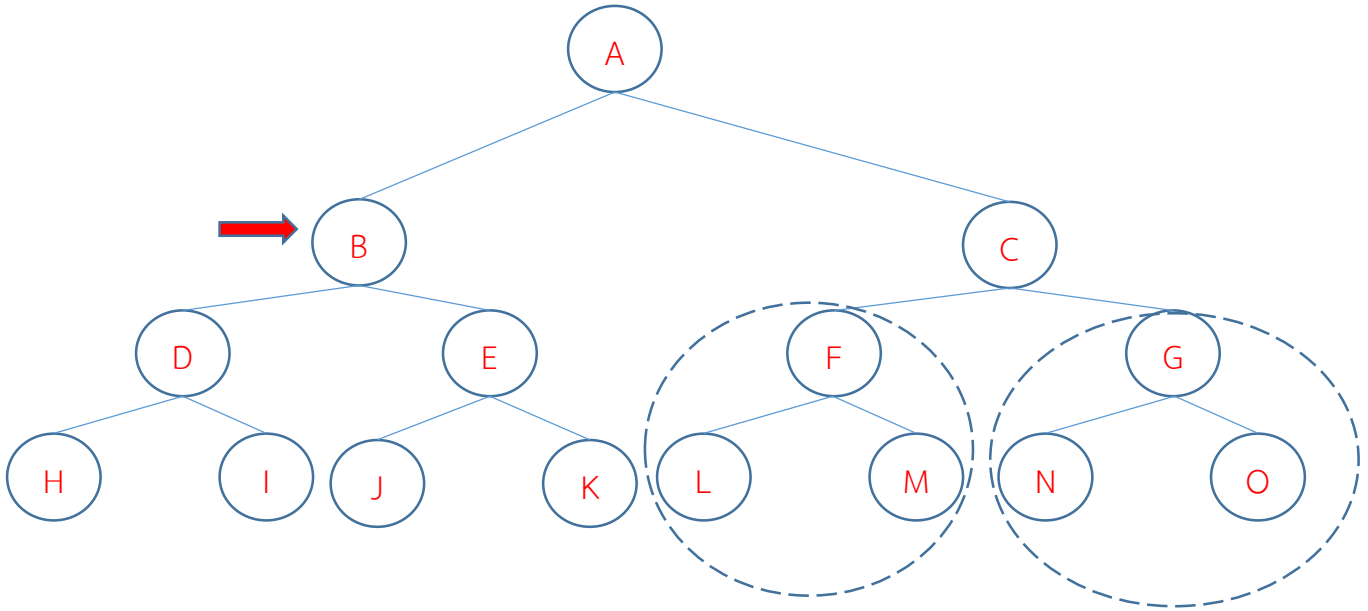
Postorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ไต่ไปทาง subtree ด้านขวาแบบ Postorder
3) ย้อนขึ้นไปยัง Root นกำ



การท่องแบบ Inorder

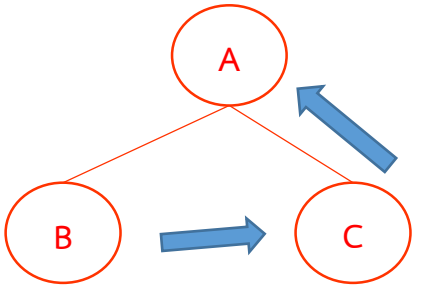
B --> C --> A



H,I,D,J,K,E,B

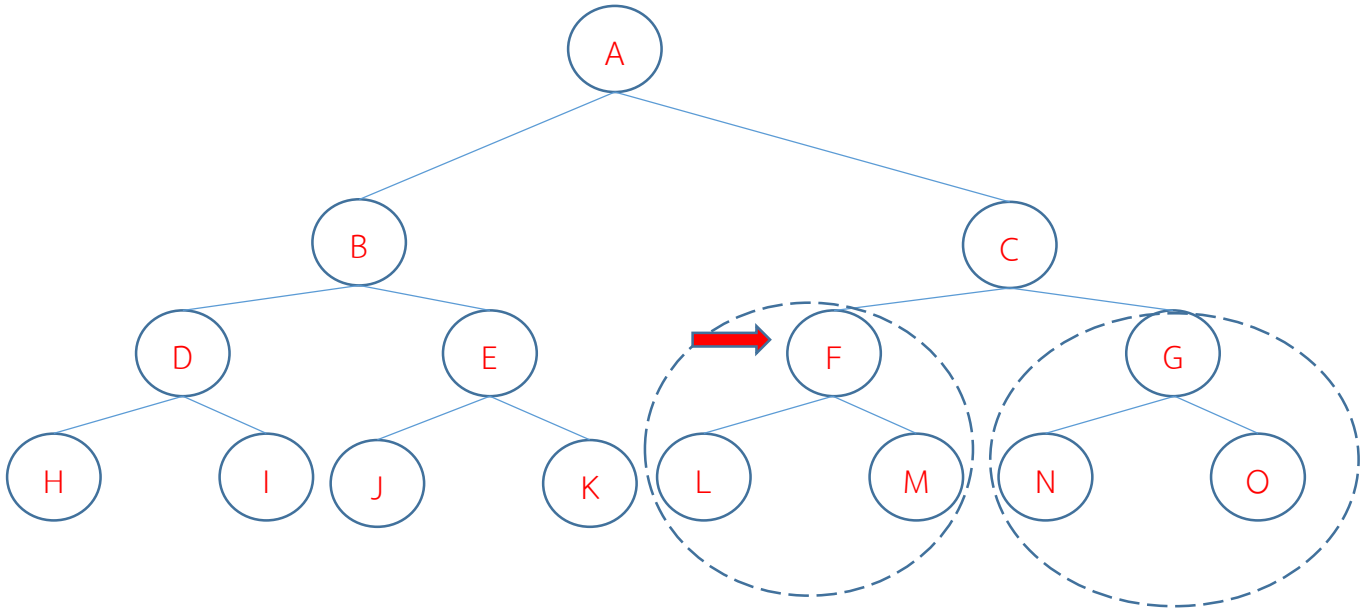
Postorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ไต่ไปทาง subtree ด้านขวาแบบ Postorder
3) ย้อนขึ้นไปยัง Root นกำ



การท่องแบบ Inorder

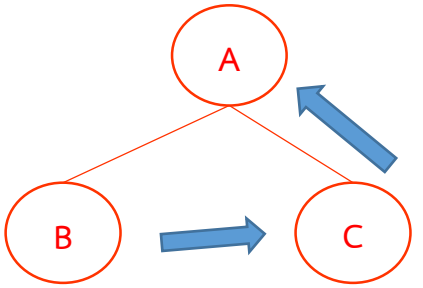
B --> C --> A



H,I,D,J,K,E,B,L,M,F

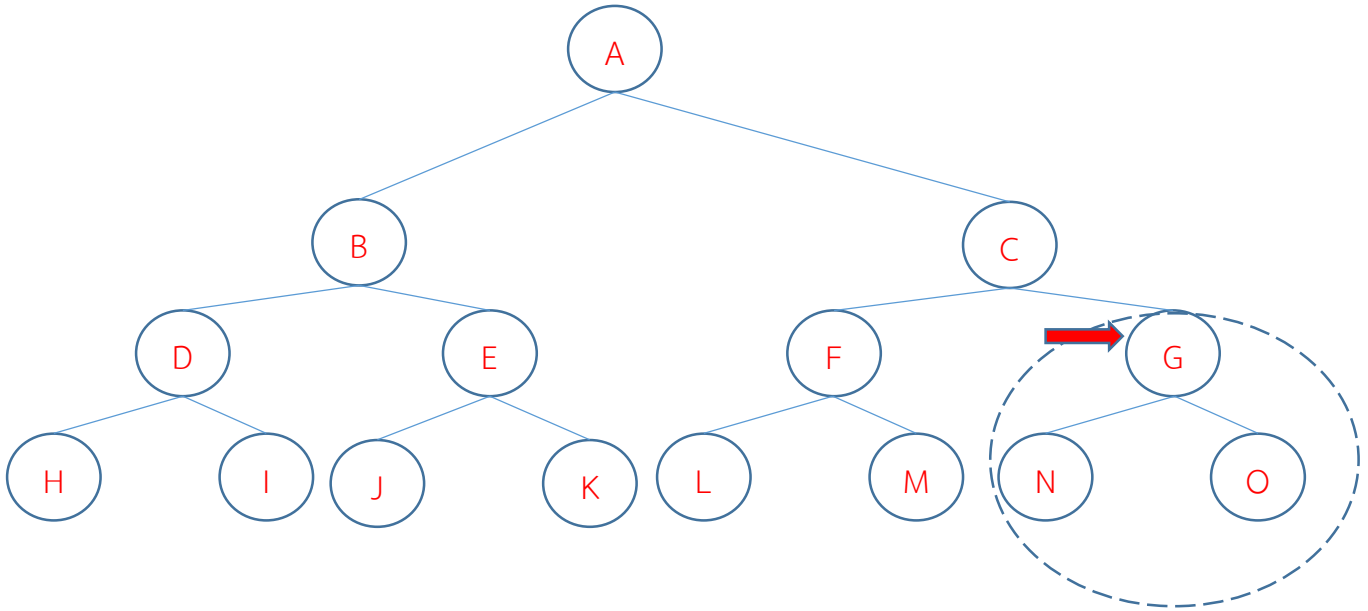
Postorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ไต่ไปทาง subtree ด้านขวาแบบ Postorder
3) ย้อนขึ้นไปยัง Root นกำ



การท่องแบบ Inorder

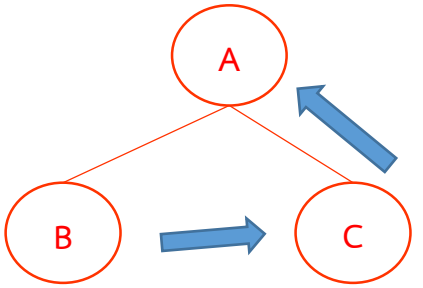
B --> C --> A



H,I,D,J,K,E,B,L,M,F,N,O,G

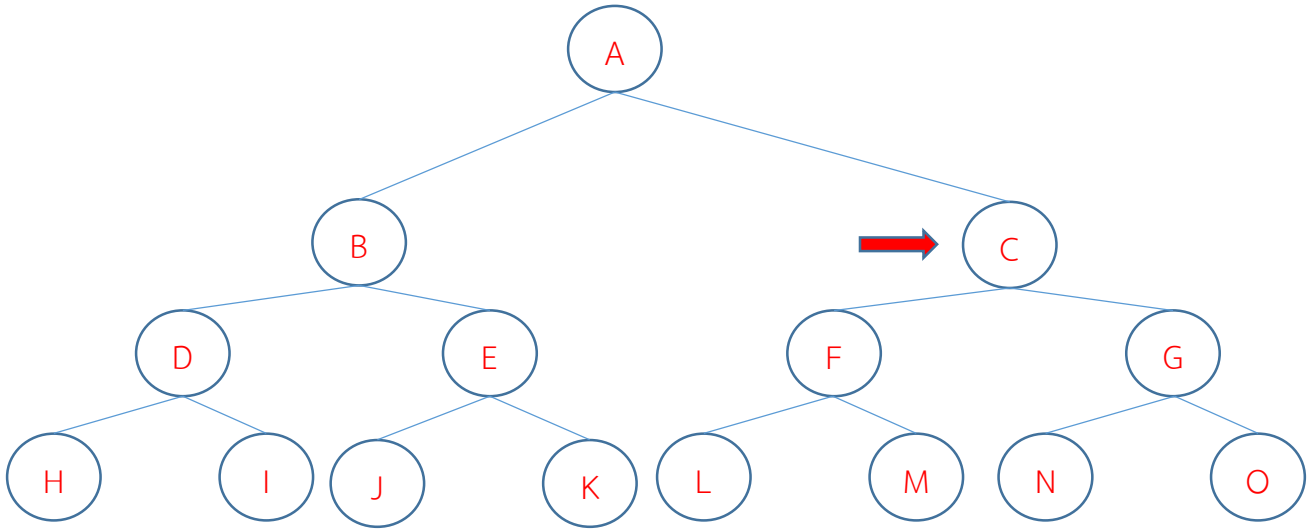
Postorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ไต่ไปทาง subtree ด้านขวาแบบ Postorder
3) ย้อนขึ้นไปยัง Root นกำ



การท่องแบบ Inorder

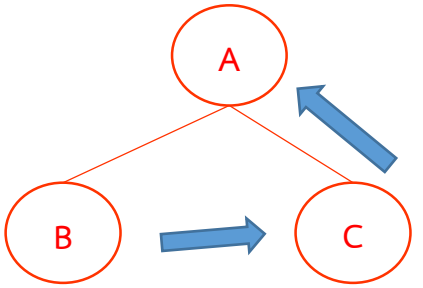
B --> C --> A



H,I,D,J,K,E,B,L,M,F,N,O,G,C

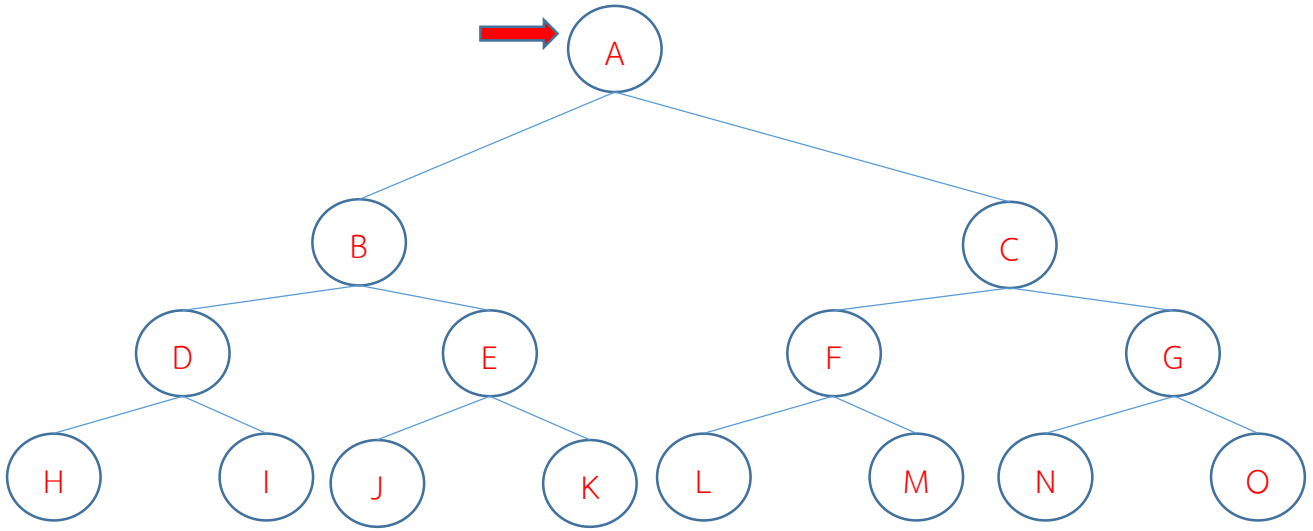
Postorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ไต่ไปทาง subtree ด้านขวาแบบ Postorder
3) ย้อนขึ้นไปยัง Root นกำ



การท่องแบบ Inorder

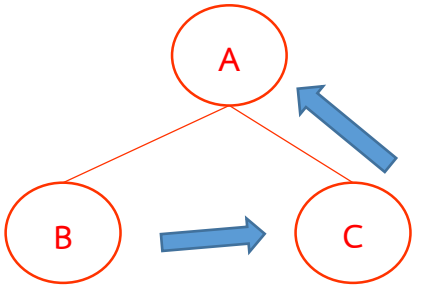
B --> C --> A



H,I,D,J,K,E,B,L,M,F,N,O,G,C,A

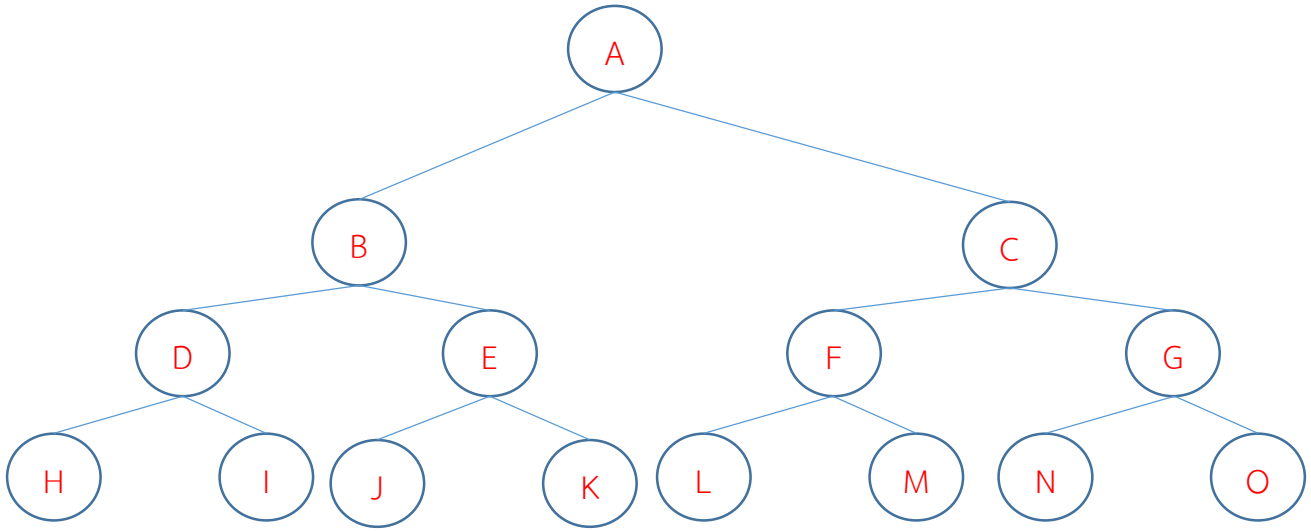
Postorder Traversal

1) เริ่มจาก subtree ด้านซ้าย
2) ไต่ไปทาง subtree ด้านขวาแบบ Postorder
3) ย้อนขึ้นไปยัง Root นกำ



การท่องแบบ Inorder

B --> C --> A

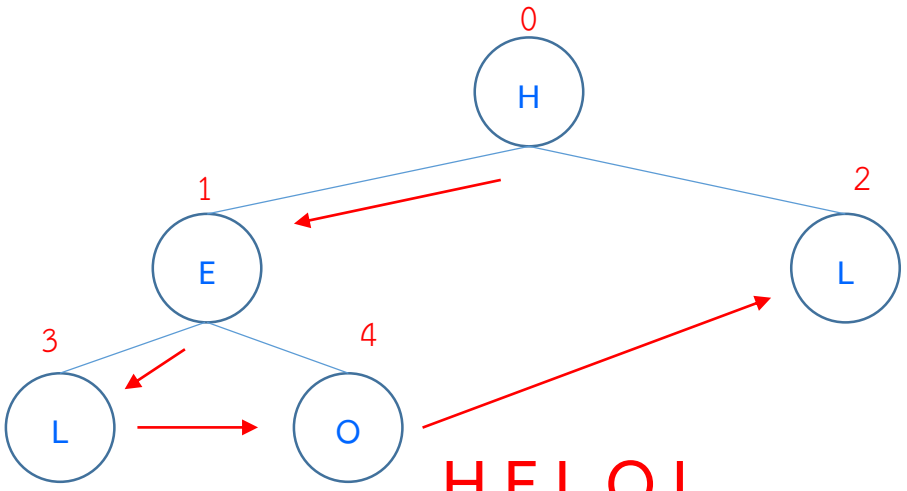


H,I,D,J,K,E,B,L,M,F,N,O,G,C,A

ขั้นตอนวิธีการท่องไปในต้นไม้

- Recursive
  - เขียนโปรแกรมง่าย สั้น แต่งง
- ใช้ Stack และ Loop
  - ไม่งง แต่โปรแกรมยาว

การท่องไปในต้นไม้แบบ Preorder แบบ Recursive



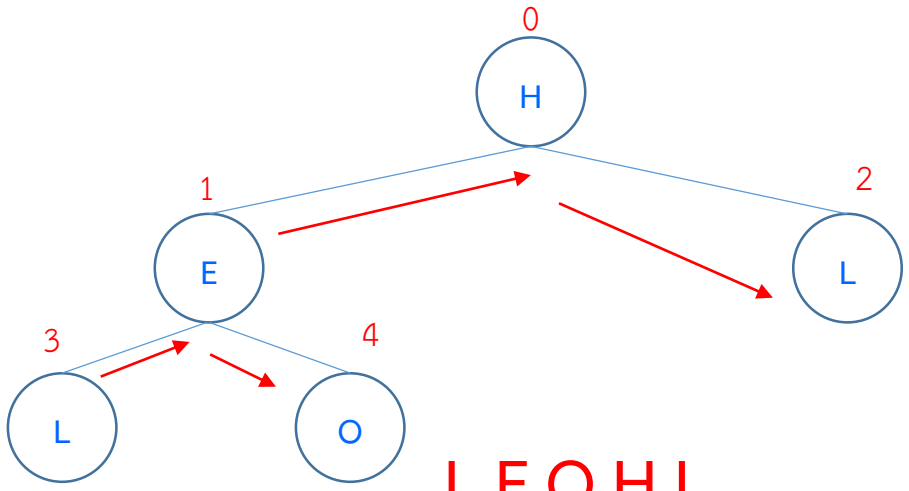
H,E,L,O,L

Index	Data	Left	Right
0	H	1	2
1	E	3	4
2	L	X	X
3	L	X	X
4	O	X	X

```c
#include<stdio.h>
char Data[]={'H','E','L','L','O'};
int Left[]= {1,3,-1,-1,-1};
int Right[]={2,4,-1,-1,-1};

void pre(int p){
if(p!=-1){
    printf("%c ",Data[p]);
    pre(Left[p]);
    pre(Right[p]);
        }
}
void preorder(void){
    pre(0);
}

main(){
 preorder();
}
```

```
H E L O L
--------------------------------
Process exited after 0.03124 seconds with
Press any key to continue . . .
```

การท่องไปในต้นไม้แบบ Inorder แบบ Recursive



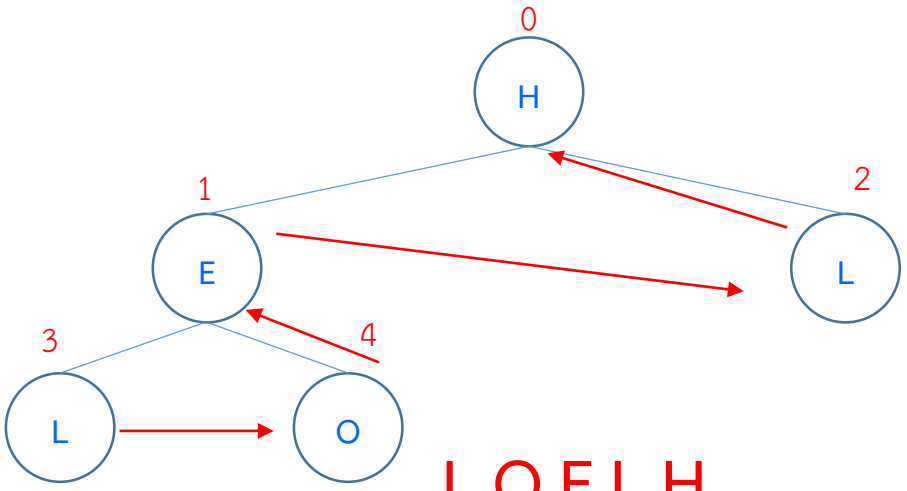
L,E,O,H,L

Index	Data	Left	Right
0	H	1	2
1	E	3	4
2	L	X	X
3	L	X	X
4	O	X	X

```c
#include<stdio.h>
char Data[]={'H','E','L','L','O'};
int Left[]= {1,3,-1,-1,-1};
int Right[]={2,4,-1,-1,-1};

void inord(p){
    if(p!=-1){
        inord(Left[p]);
        printf("%c ",Data[p]);
        inord(Right[p]);
    }
}
void inorder()
{
    inord(0);
}


main(){
 inorder();
}
```

```
L E O H L
--------------------------------
Process exited after 0.0118 seconds with return
Press any key to continue . . .
```

การท่องไปในต้นไม้แบบ Postorder แบบ Recursive



L,O,E,L,H

Index	Data	Left	Right
0	H	1	2
1	E	3	4
2	L	X	X
3	L	X	X
4	O	X	X

```c
#include<stdio.h>
char Data[]={'H','E','L','L','O'};
int Left[]= {1,3,-1,-1,-1};
int Right[]={2,4,-1,-1,-1};

void postorder(){
    post(0);
}
void post(p){
    if(p!=-1){
        post(Left[p]);
        post(Right[p]);
        printf("%c ",Data[p]);
    }
}
main(){
 postorder();
}
```

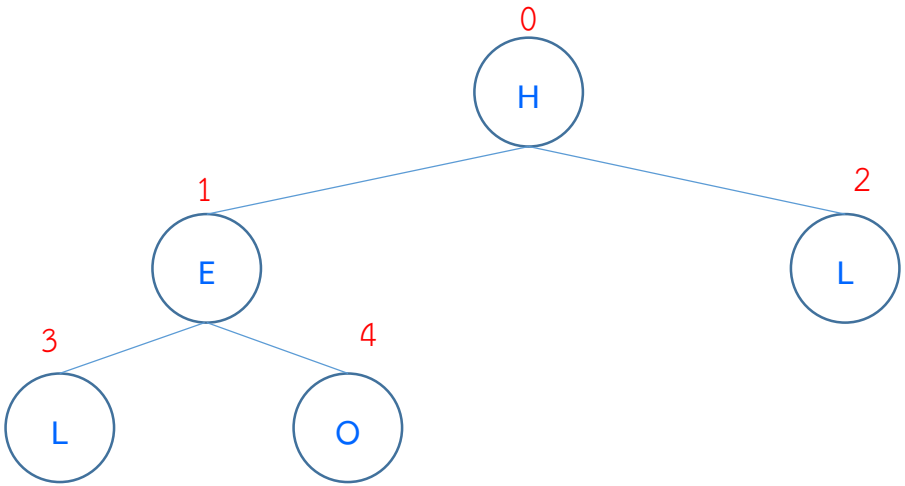
```
G:\My Drive\Lectures\CS221\slides\treearray_traverse.exe
L O E L H
--------------------------------
Process exited after 0.01352 seconds
Press any key to continue . . .
```

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

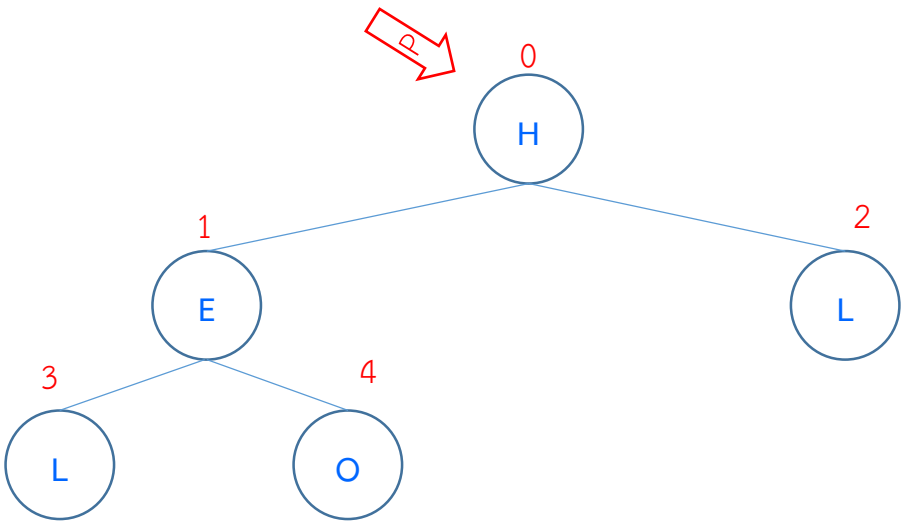
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

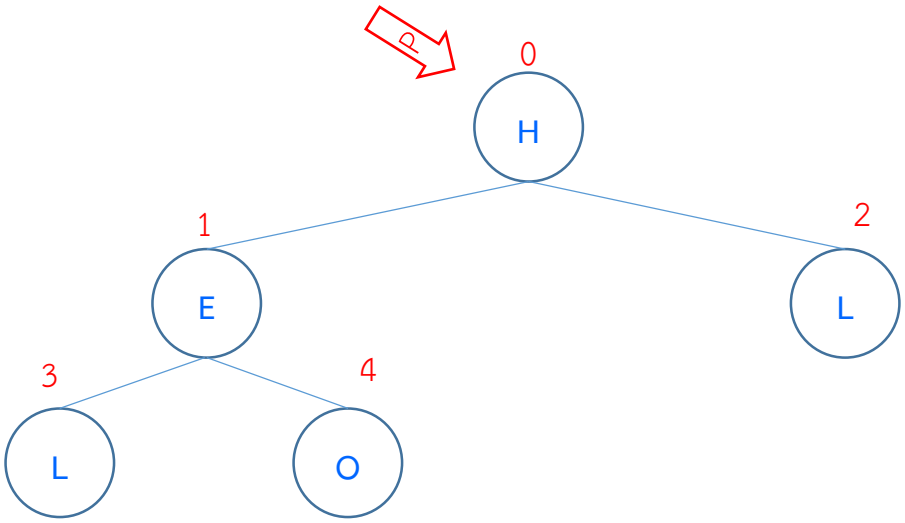
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

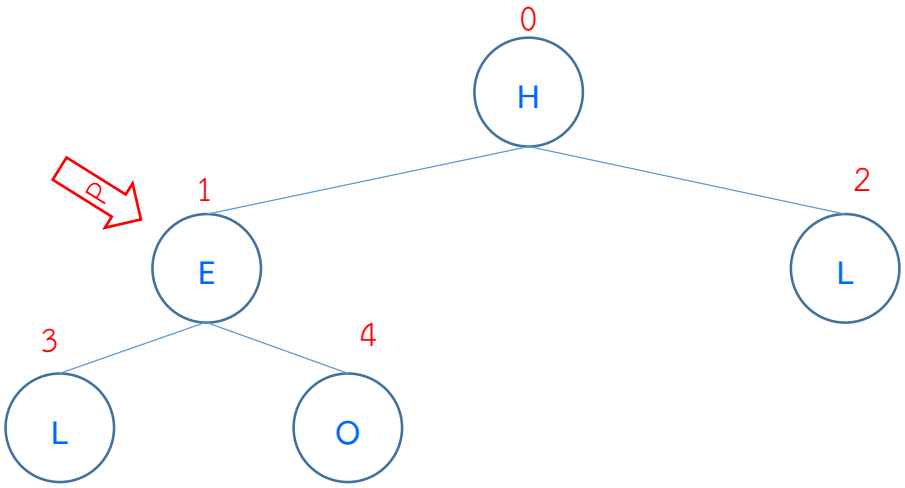
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

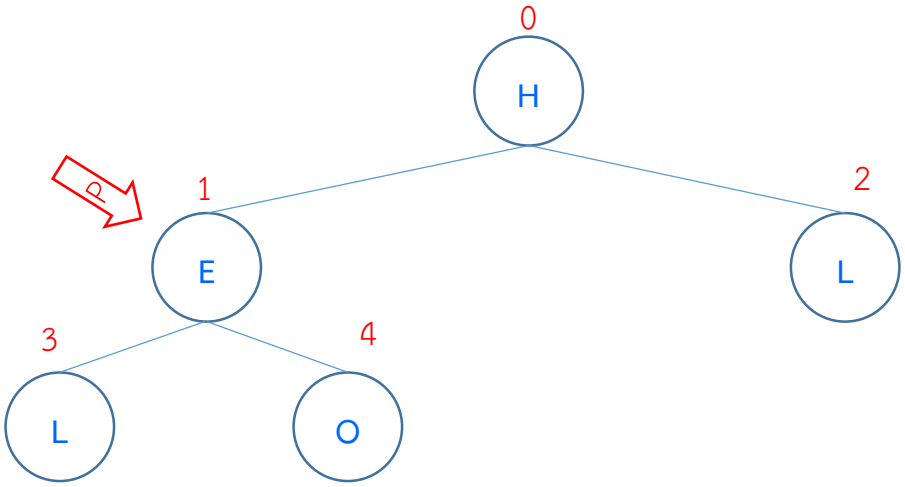
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

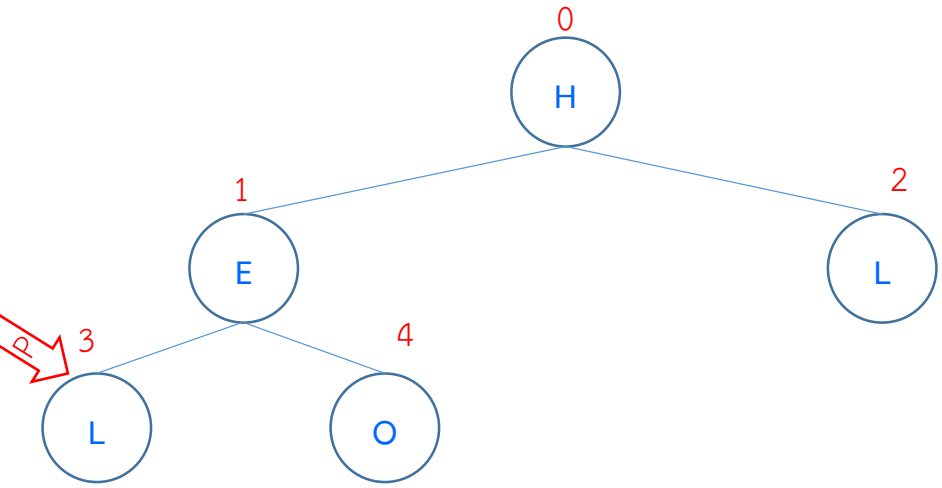
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

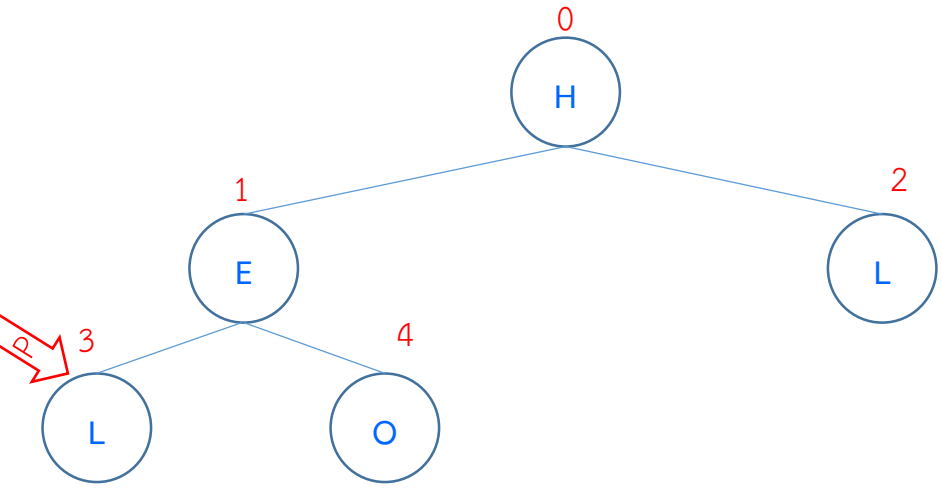
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

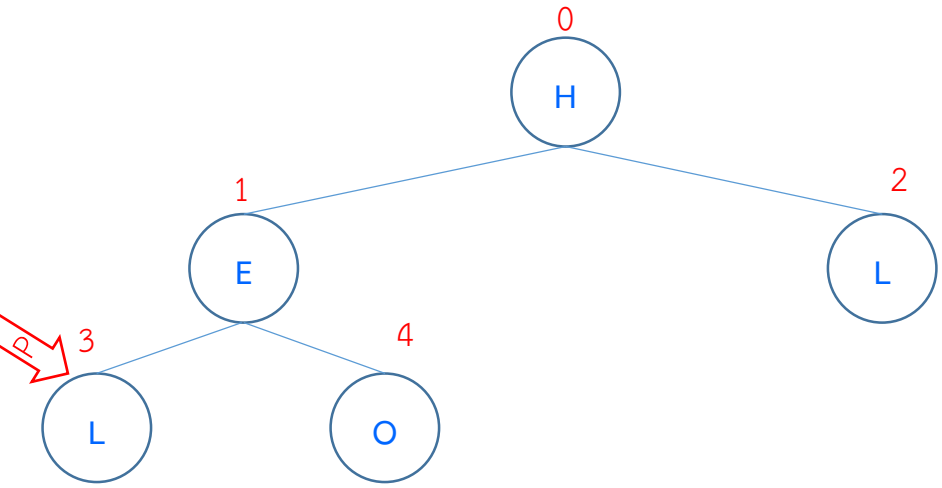
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

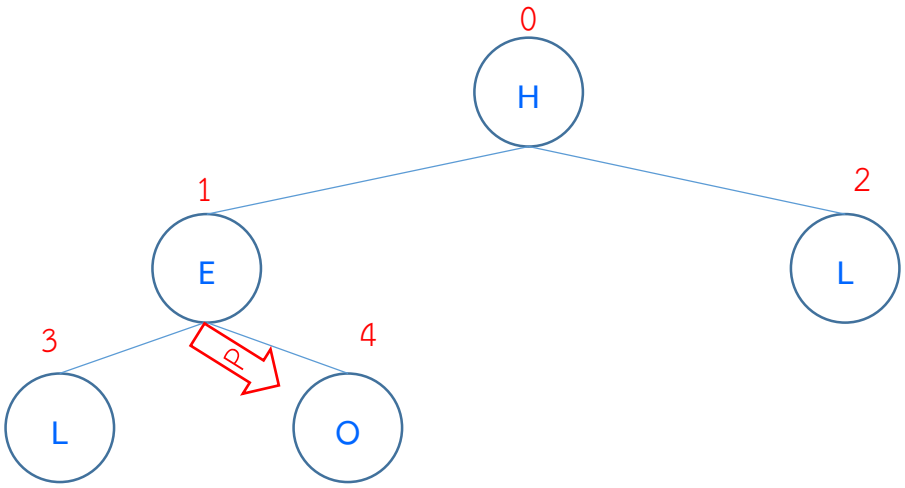
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

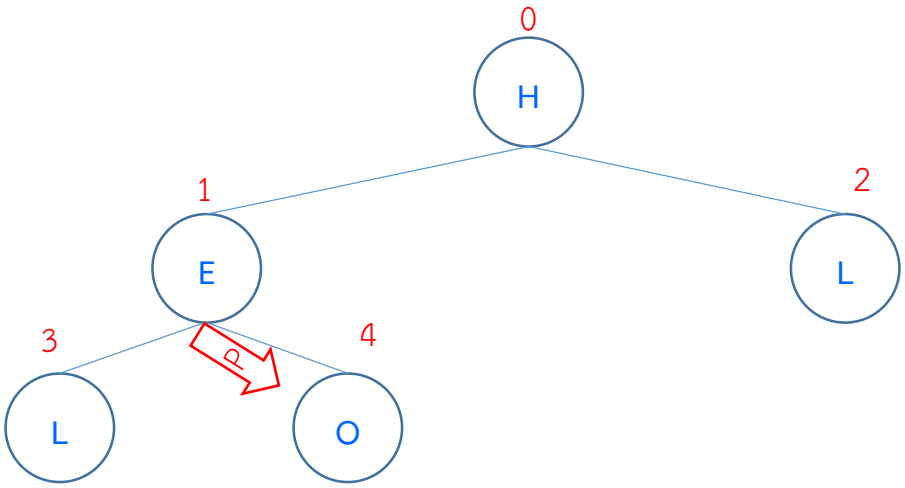
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L O

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

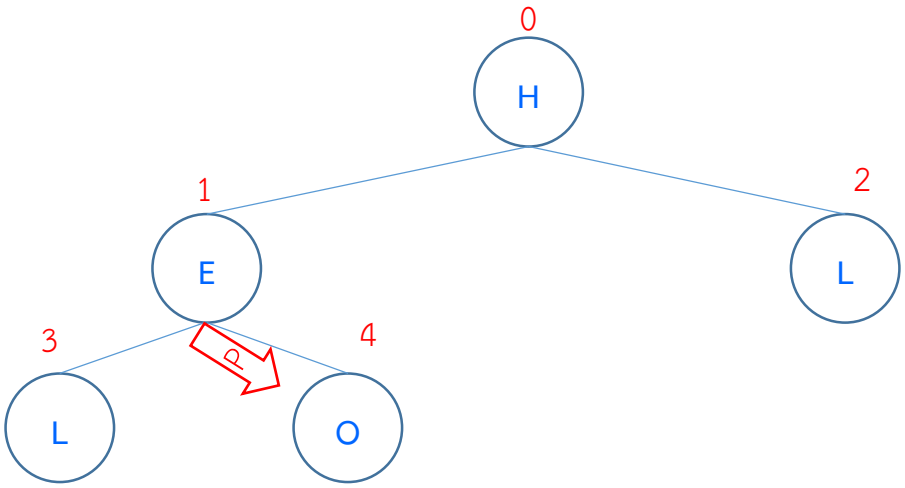
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L O

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

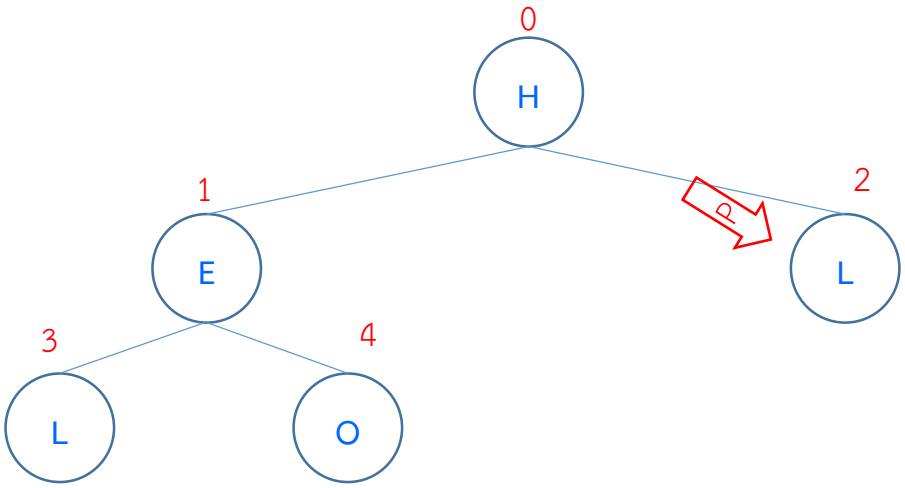
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L O
2	ว่าง	H E L O

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

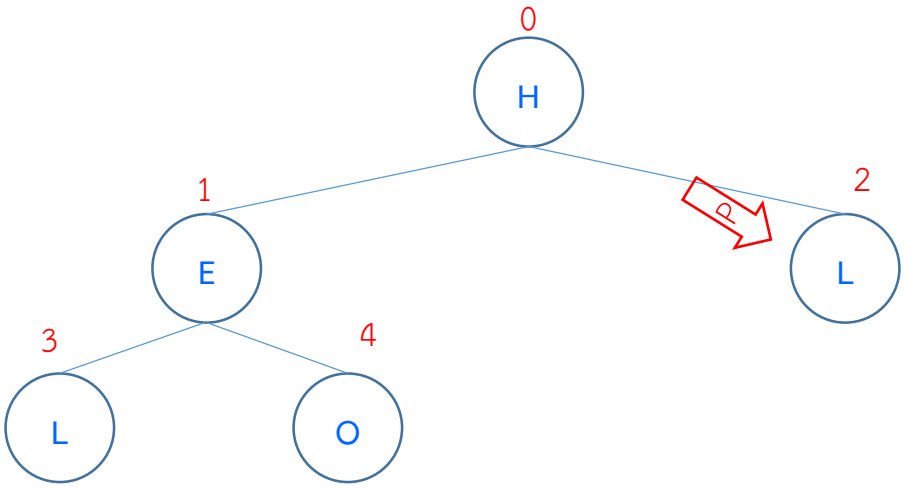
1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L O
2	ว่าง	H E L O

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

1) push ตำแหน่งของ root node
2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
4) pop มาค่าใส่ p



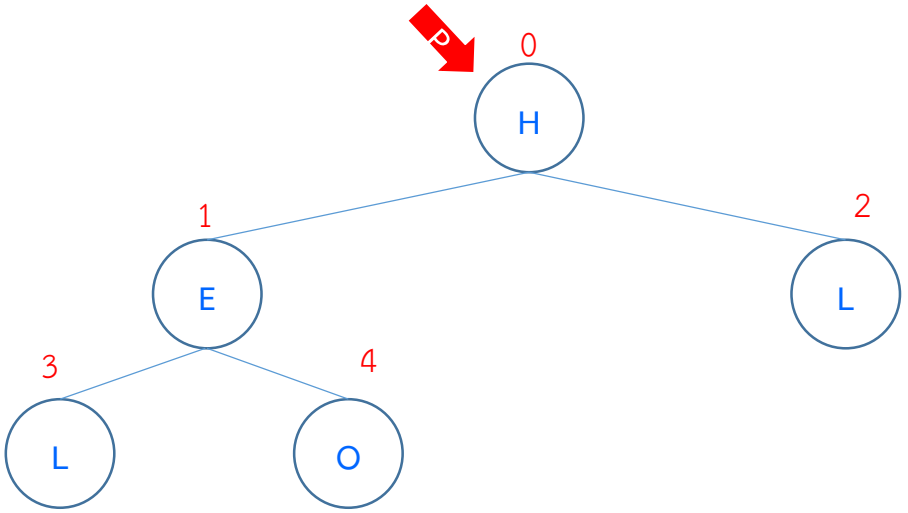
p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L O
2	ว่าง	H E L O
2	ว่าง	H E L O L

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

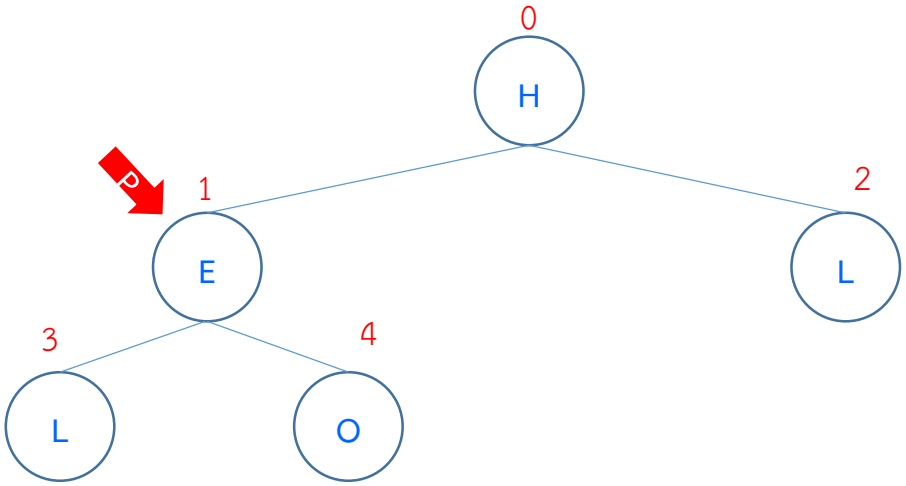
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0	ว่าง	

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

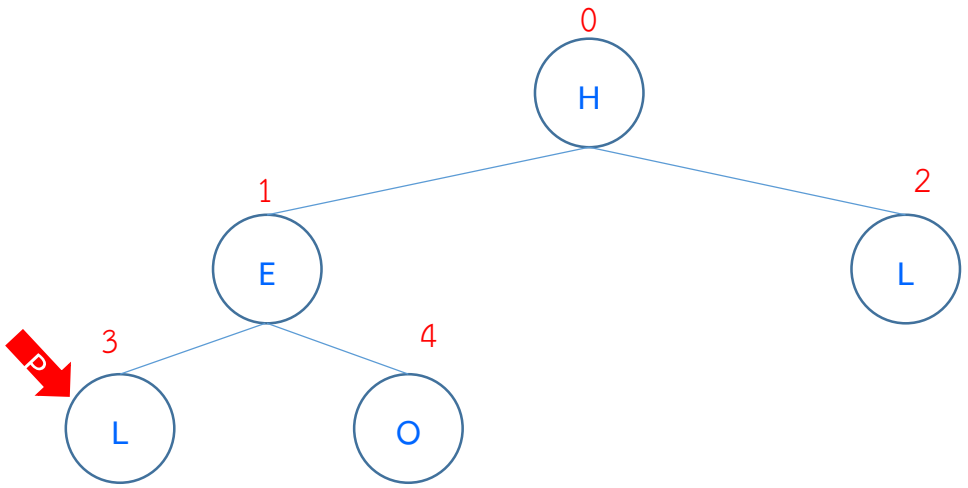
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

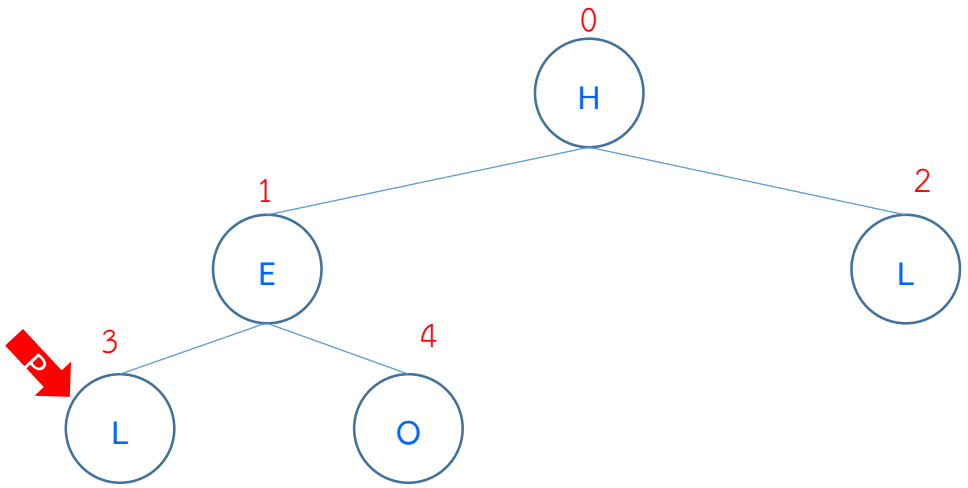
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

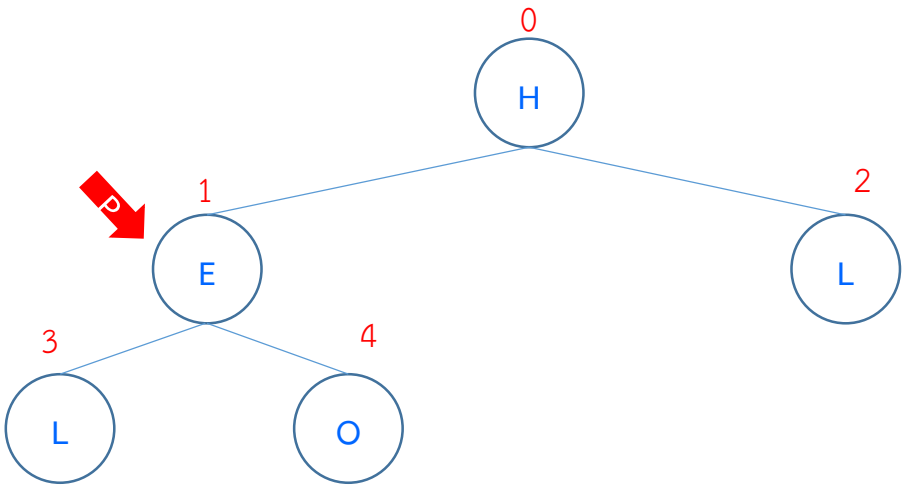
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

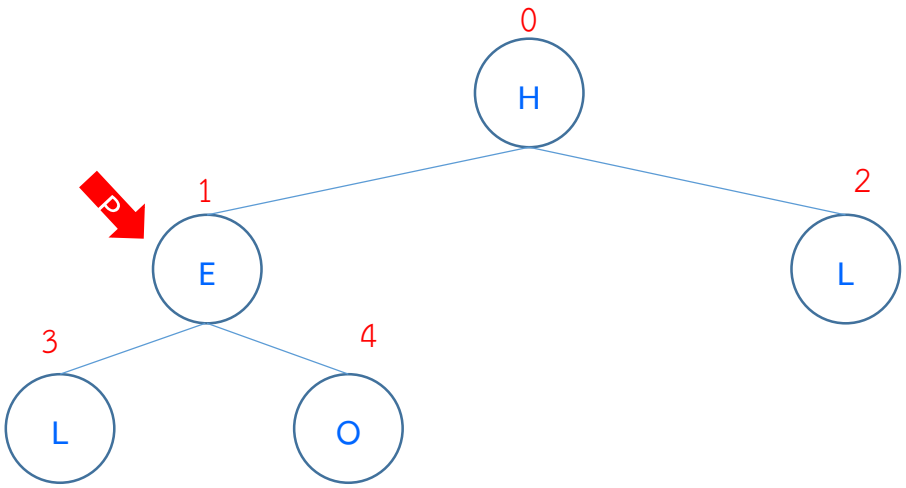
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

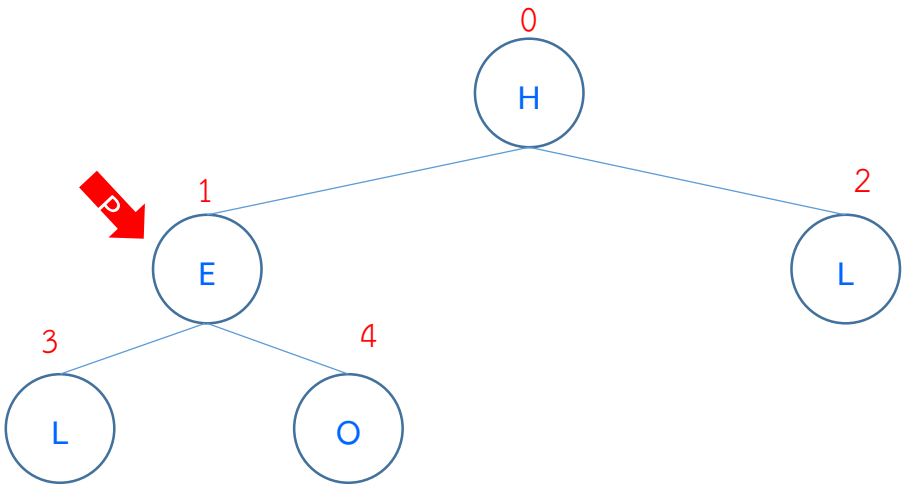
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามีกิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

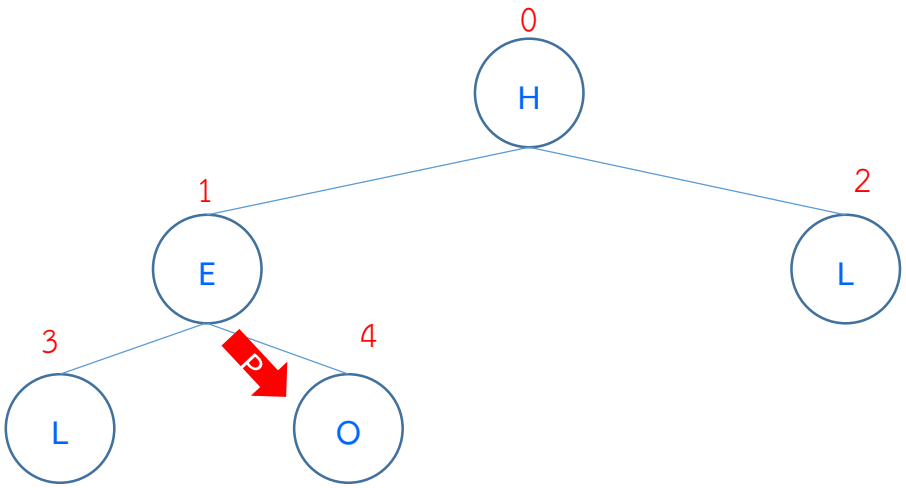
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

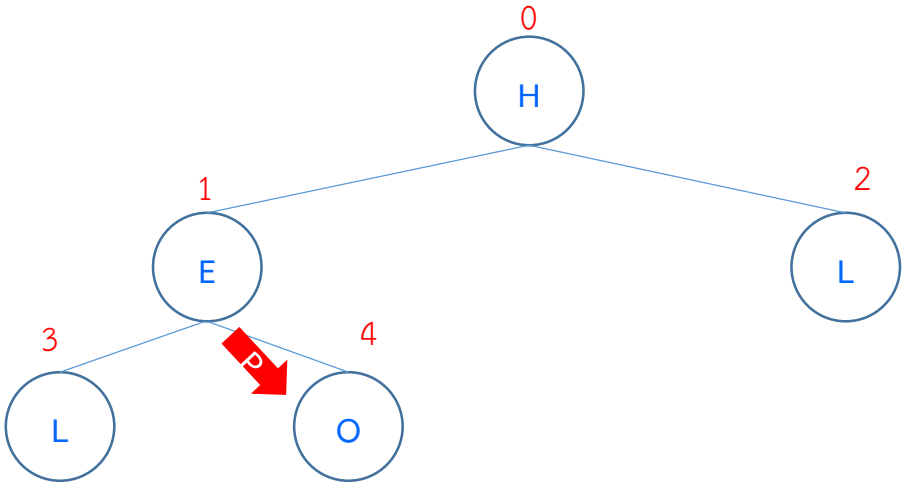
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E
4	0	L E O

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

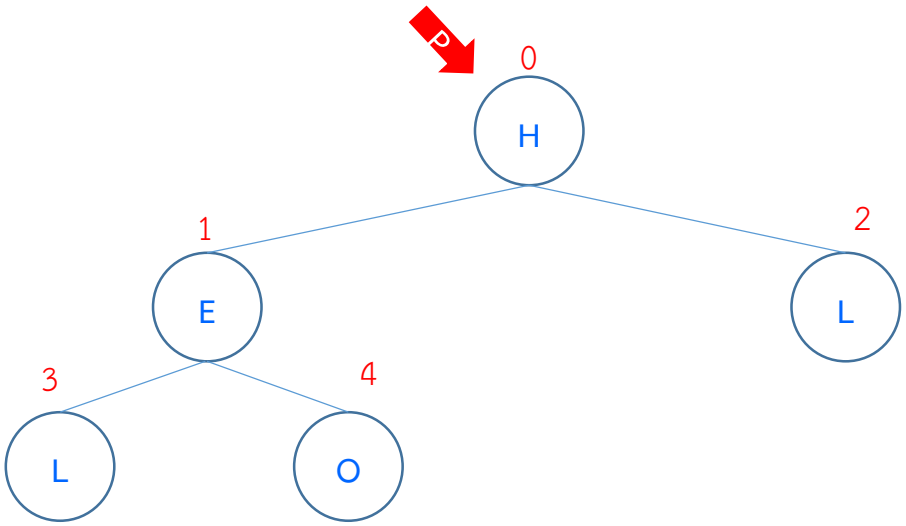
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E
4	0	L E O
0		L E O

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

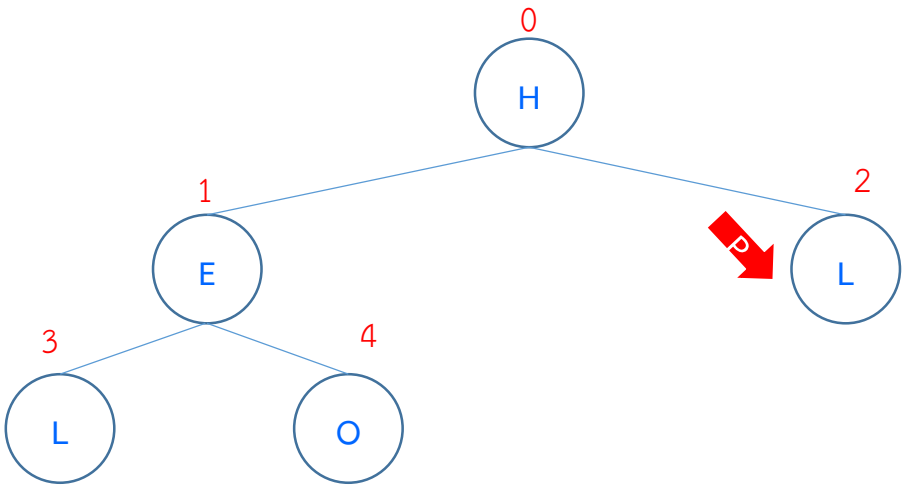
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E
4	0	L E O
0		L E O
0		L E O H

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

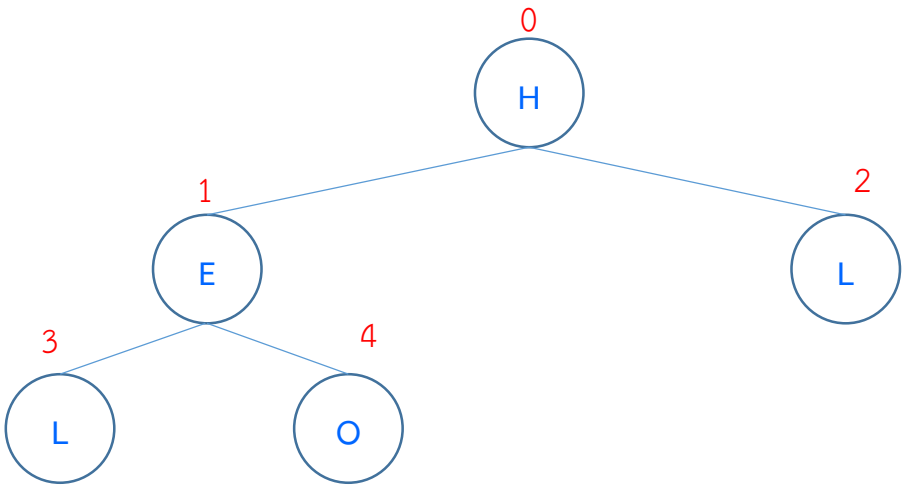
1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E
4	0	L E O
0		L E O
0		L E O H
2		L E O H L

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

1) กำหนด p คือตำแหน่งของ root node
2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E
4	0	L E O
0		L E O
0		L E O H
2		L E O H L

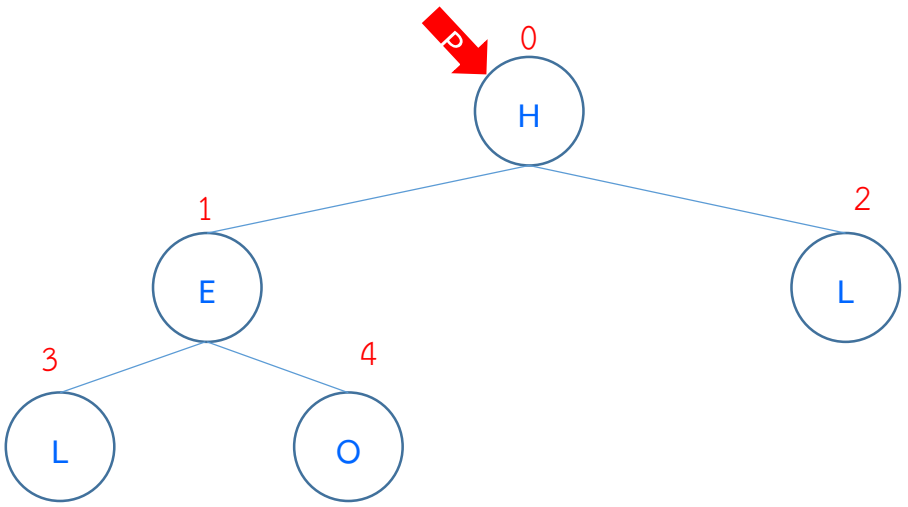
การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack
ยากกว่า Preorder และ Inorder เพราะขั้นตอนการทำงานของ กิ่งซ้ายและขวาจะไม่เหมือนกัน

การเก็บ node ขวาจึงเพิ่ม $ ไว้ด้านหน้าตำแหน่ง เพื่อให้รู้ว่าคือ node ขวา

1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

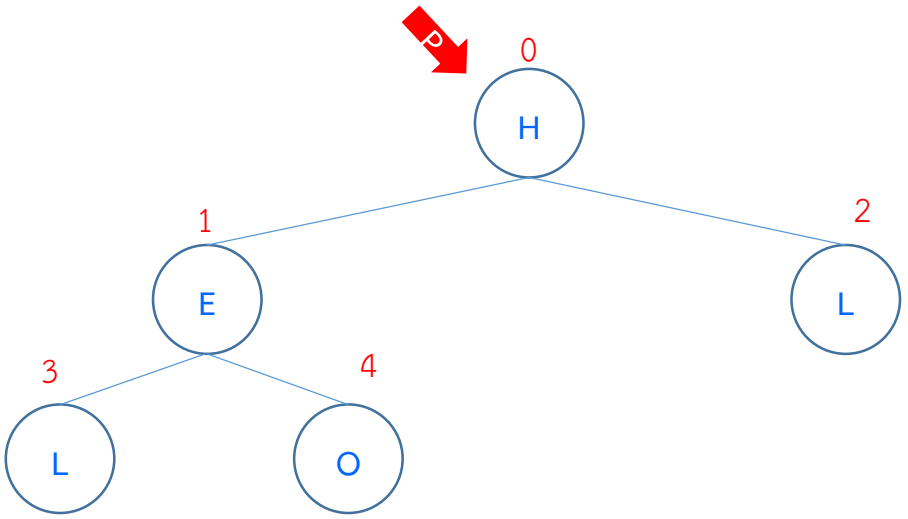
1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

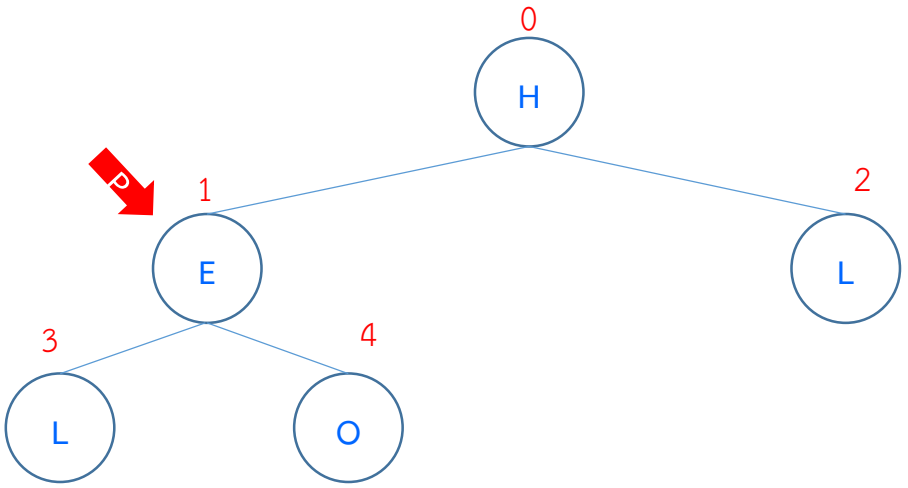
1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
0	$2 0	

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

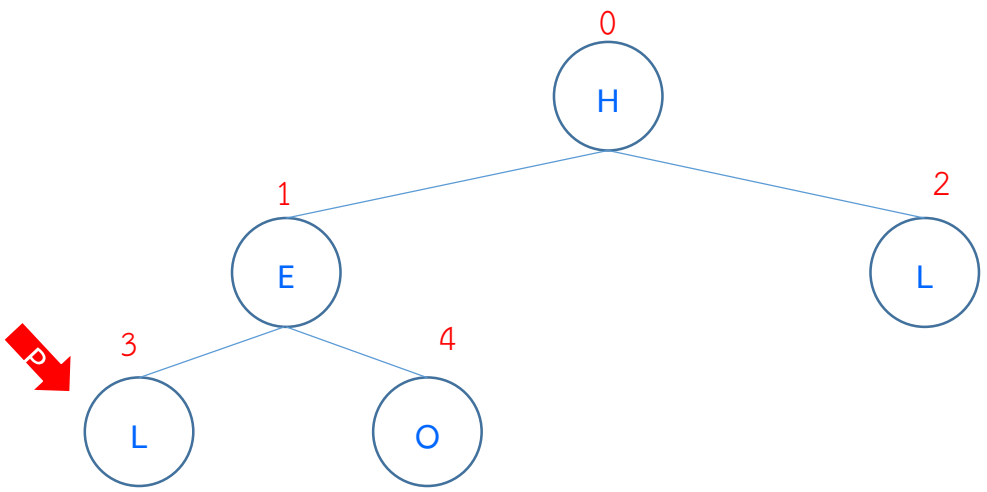
1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
0	$2 0	
1	$2 0	
1	$4 1 $2 0	

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

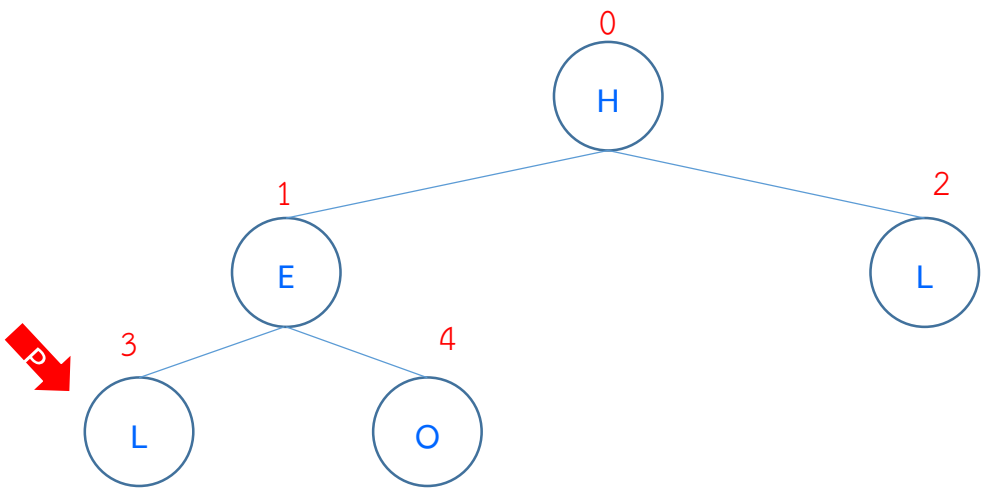
1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
0	$2 0	
1	$2 0	
1	$4 1 $2 0	
3	$4 1 $2 0	

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

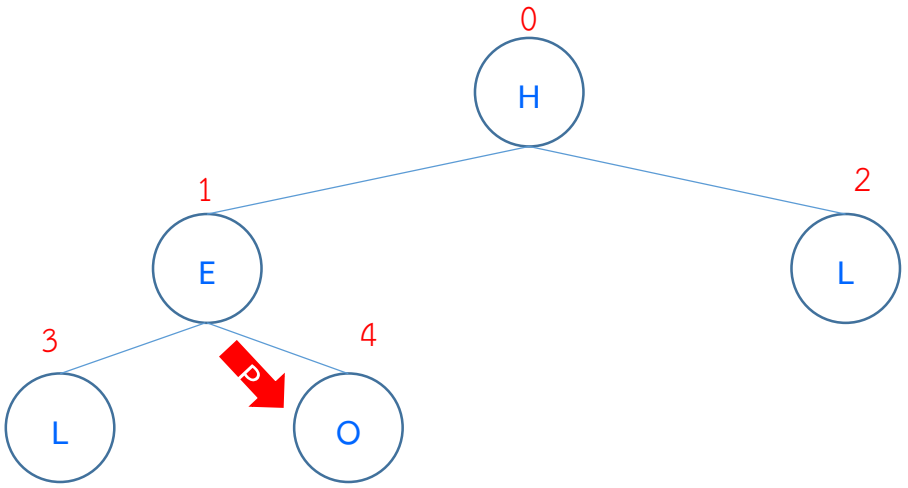
1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
0	$2 0	
1	$2 0	
1	$4 1 $2 0	
3	$4 1 $2 0	L

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

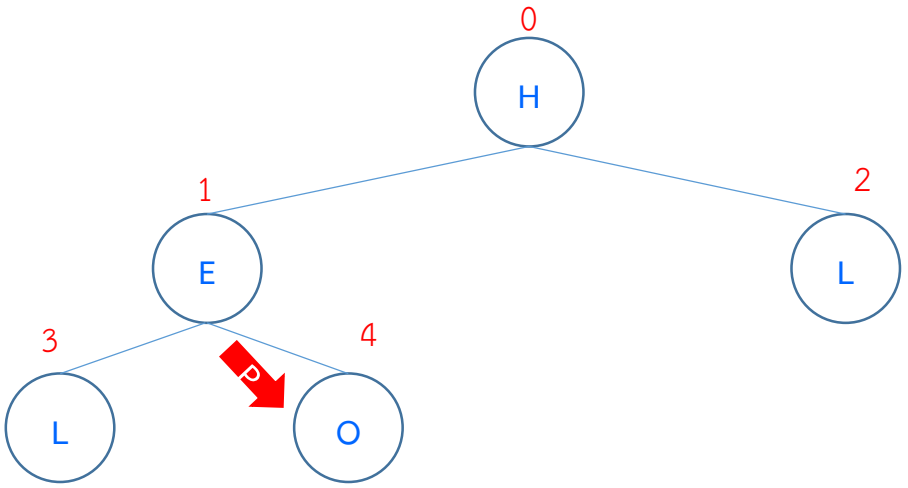
1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
0	$2 0	
1	$2 0	
1	$4 1 $2 0	
3	$4 1 $2 0	L
$4	1 $2 0	

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

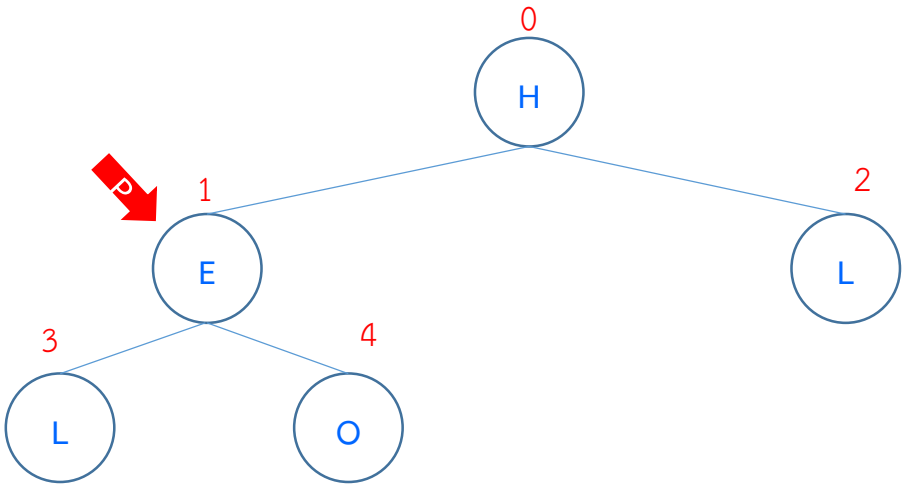
1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
0	$2 0	
1	$2 0	
1	$4 1 $2 0	
3	$4 1 $2 0	L
$4	1 $2 0	L O

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

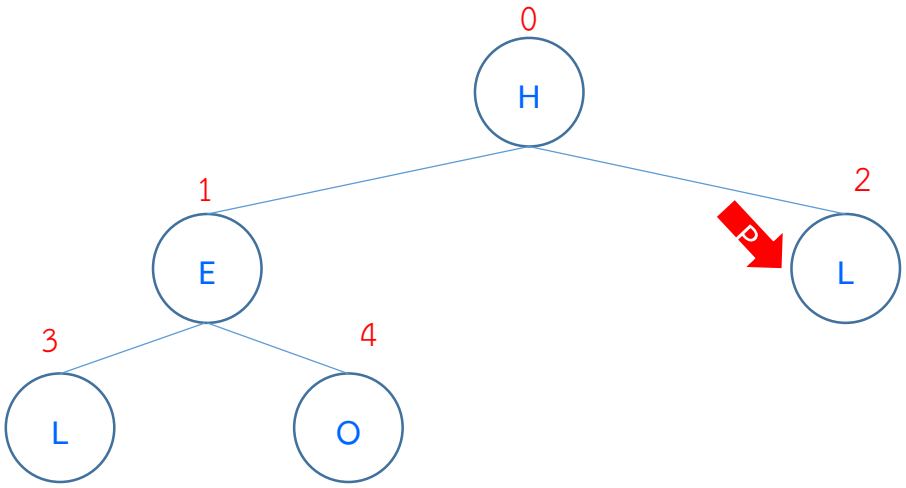
1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
0	$2 0	
1	$2 0	
1	$4 1 $2 0	
3	$4 1 $2 0	L
$4	1 $2 0	L O
1	0	L O E

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

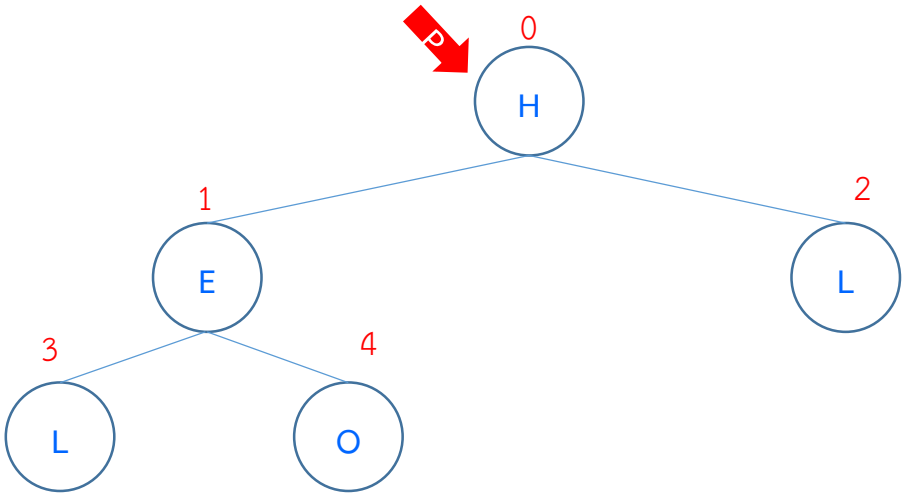
1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
0	$2 0	
1	$2 0	
1	$4 1 $2 0	
3	$4 1 $2 0	L
$4	1 $2 0	L O
1	1 0	L O E
$2	0	L O E L

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

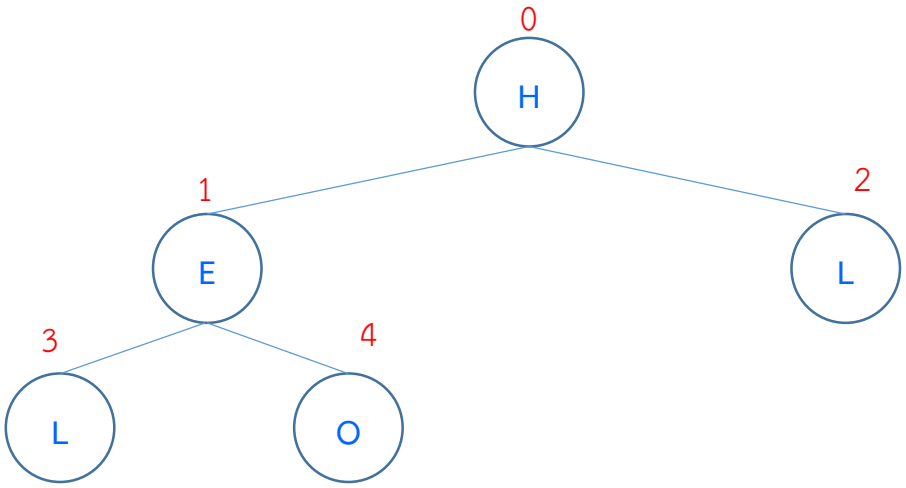
1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
0	$2 0	
1	$2 0	
1	$4 1 $2 0	
3	$4 1 $2 0	L
$4	1 $2 0	L O
1	0	L O E
$2	0	L O E L
0		L O E L

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำจนกว่า stack จะว่าง
2) ไต่ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้ push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่ $ ไว้ด้านหน้า
3) pop ตำแหน่งที่ไม่มี $ ออกมาทั้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี $ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2



p	Stack	Node ที่ผ่าน
0		
0	$2 0	
1	$2 0	
1	$4 1 $2 0	
3	$4 1 $2 0	L
$4	1 $2 0	L O
1	0	L O E
$2	0	L O E L
		L O E L H

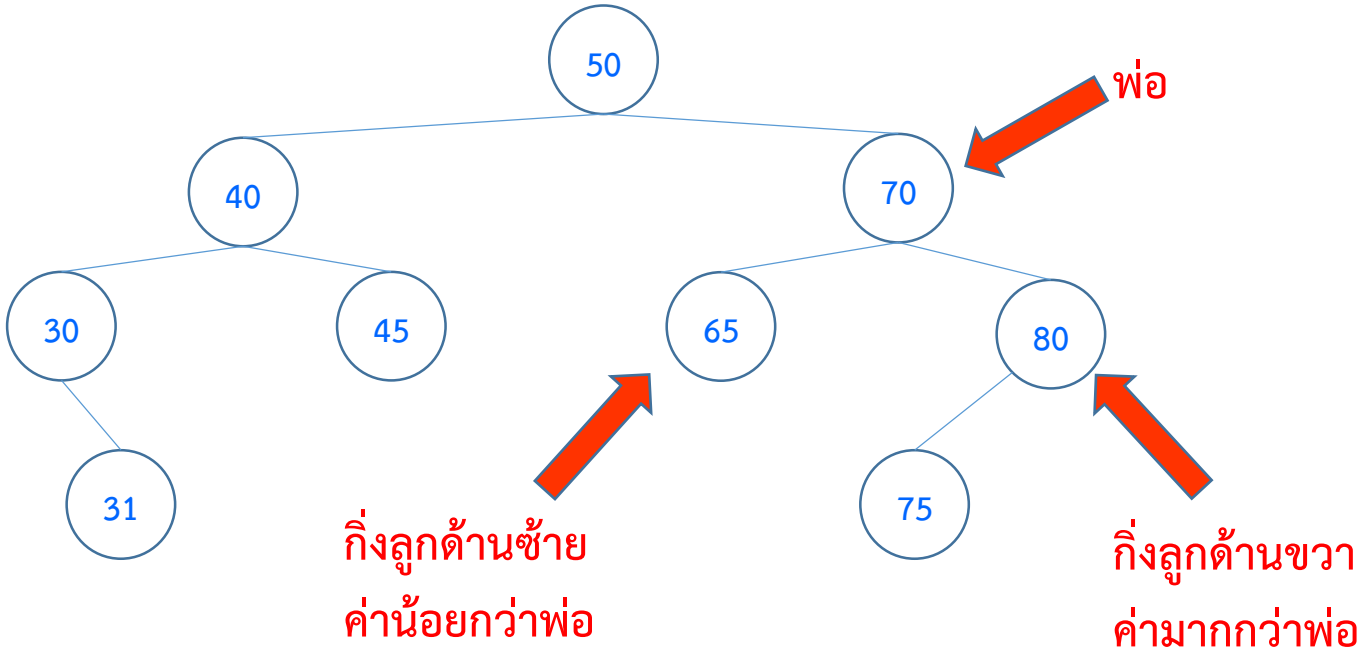
การสร้าง เพิ่ม หรือลบ ข้อมูลออกจากต้นไม้
จำเป็นต้อง รู้วิธีการ Traverse ก่อน จึงจะสามารถ ทำ operation เหล่านี้ได้

ย้อนกลับไปยังต้นไม่ทวิภาค ต้นแรกในบทนี้

Binary Search Tree

Binary Search Trees

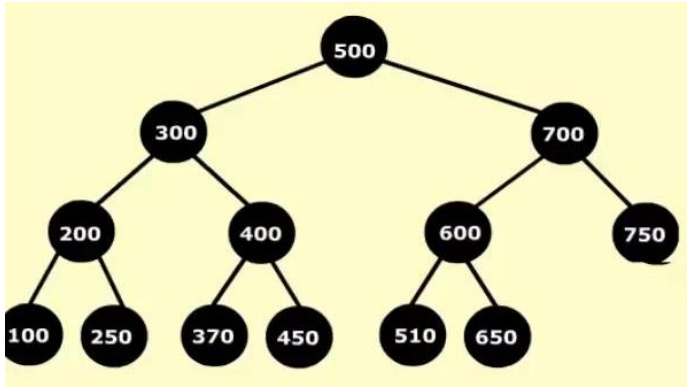
- ค่าของ node จะซ้ำกันไม่ได้
- กิ่งลูกทางด้านซ้ายจะต้องมีค่าน้อยกว่าพ่อ และกิ่งลูกทางด้านขวาต้องมีค่ามากกว่าพ่อเสมอ
- ข้อมูลในต้นไม้ต้องเป็นแบบที่สามารถเปรียบเทียบกันได้เท่านั้น

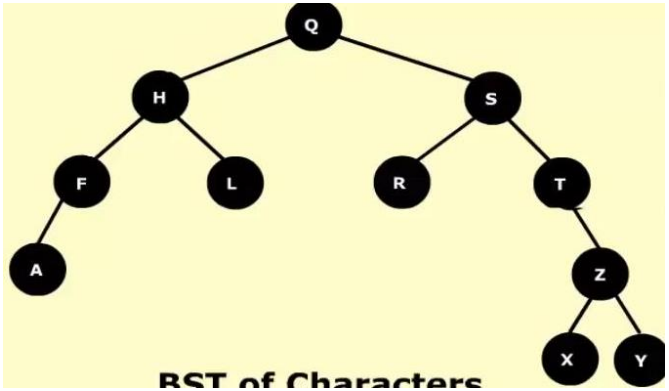


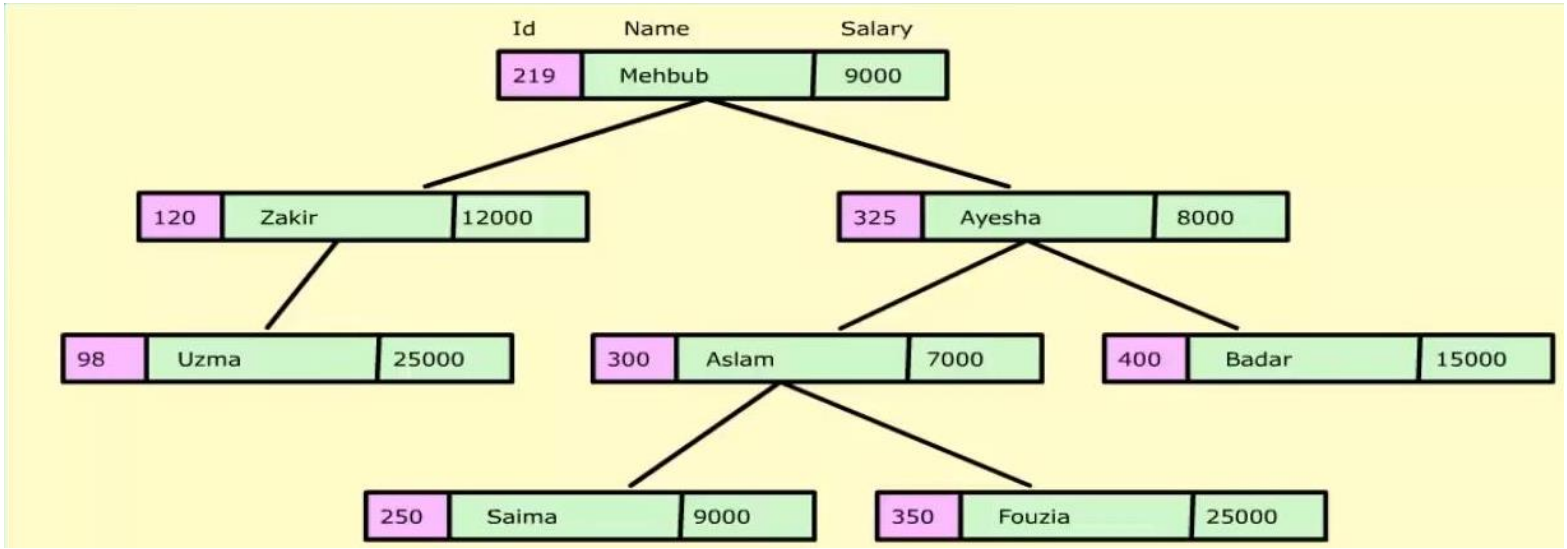
ข้อมูลไม่ซ้ำกัน

ข้อมูลในแต่ละ node อาจเป็นจำนวน หรือ ตัวอักษรก็ได้

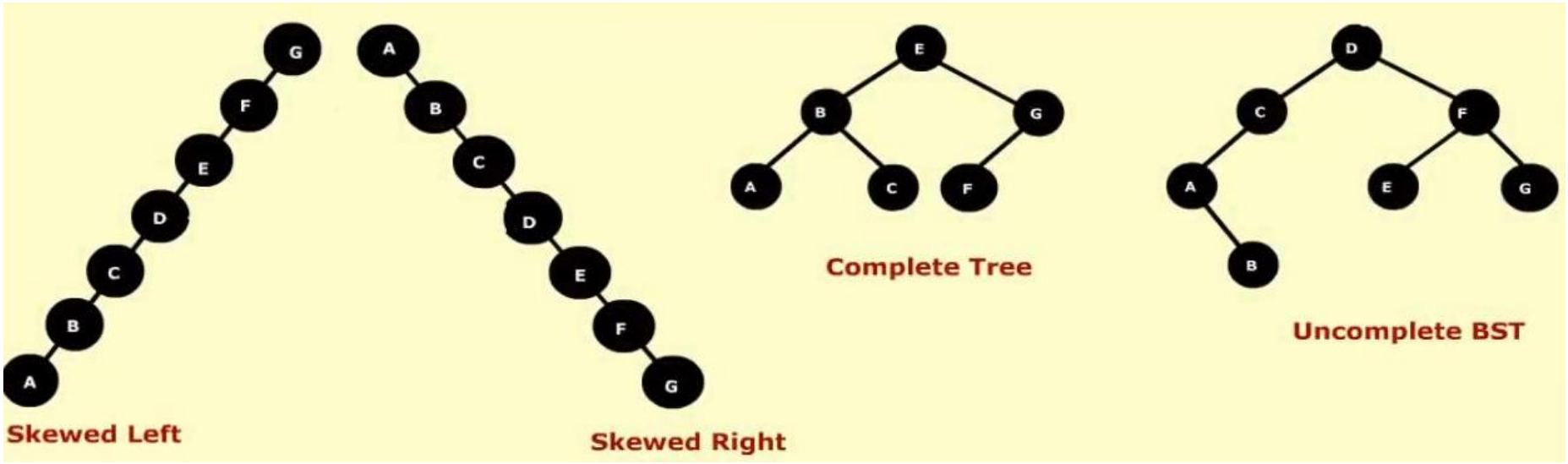
หากต้องการเก็บข้อมูลที่ซับซ้อนกว่านั้น ให้สร้าง Structure และกำหนด ID ของ Node



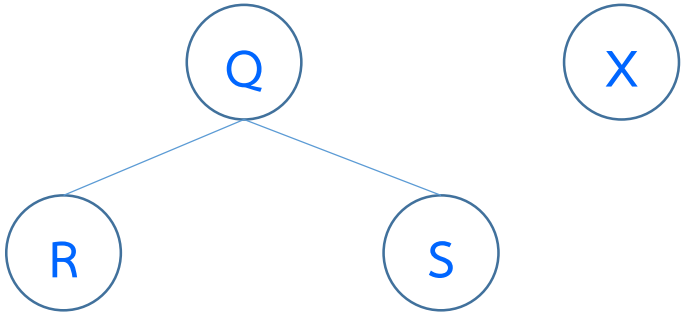




ข้อมูลเดียวกัน แต่ลำดับไม่เหมือนกัน จะทำให้เกิดต้นไม้ที่ต่างกัน



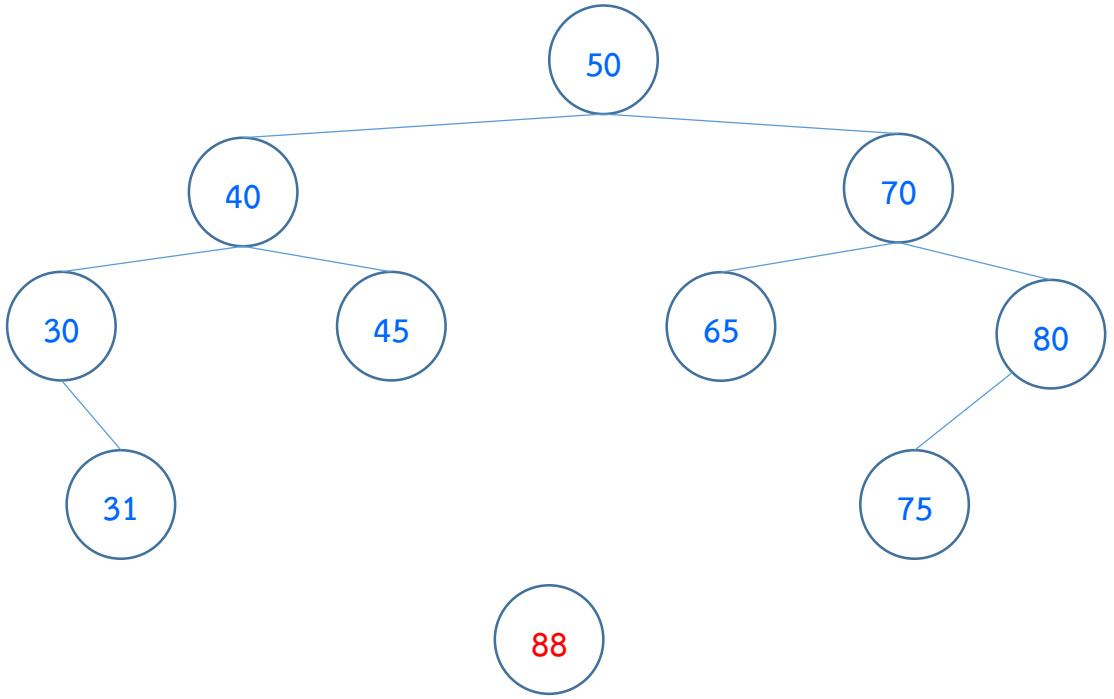
การเพิ่มข้อมูลใน Binary Search Tree



1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
2) หากค่าน้อยกว่า Root ให้ไต่ไปทางซ้าย หากมากกว่าให้ไต่ไปทางขวา
3) ทำซ้ำไปเรื่อย ๆจนกว่าจะถึง node สุดท้าย
4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา

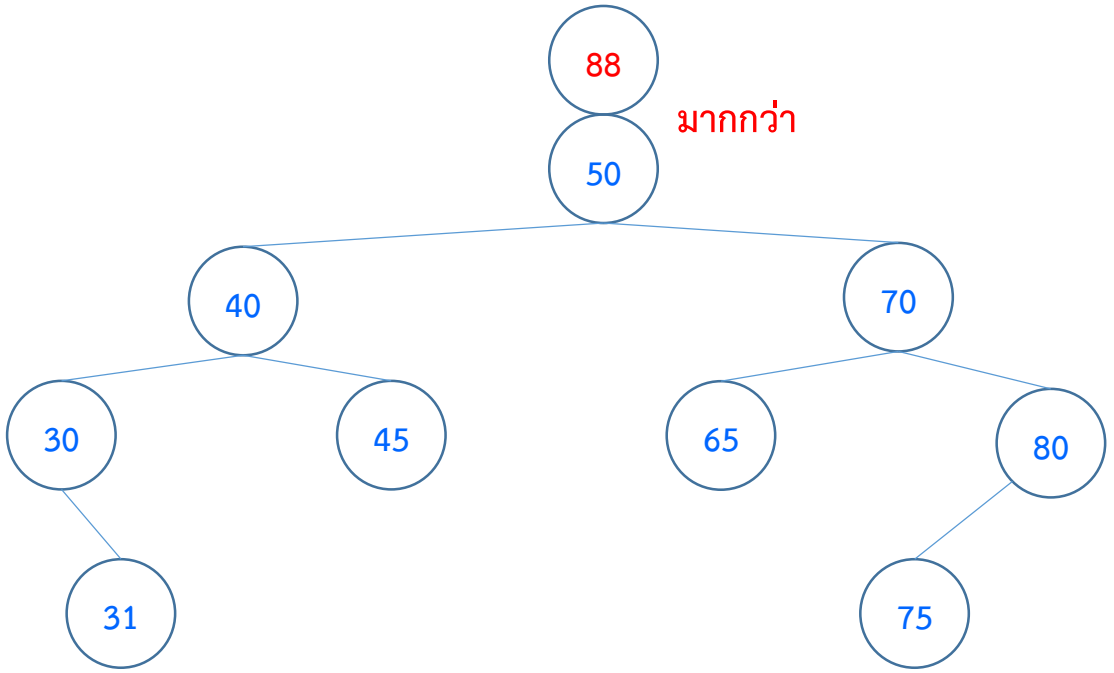
การเพิ่มข้อมูลใน Binary Search Tree

1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
2) หากค่าน้อยกว่า Root ให้ไต่ไปทางซ้าย หากมากกว่าให้ไต่ไปทางขวา
3) ทำซ้ำไปเรื่อย ๆจนกว่าจะถึง node สุดท้าย
4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา



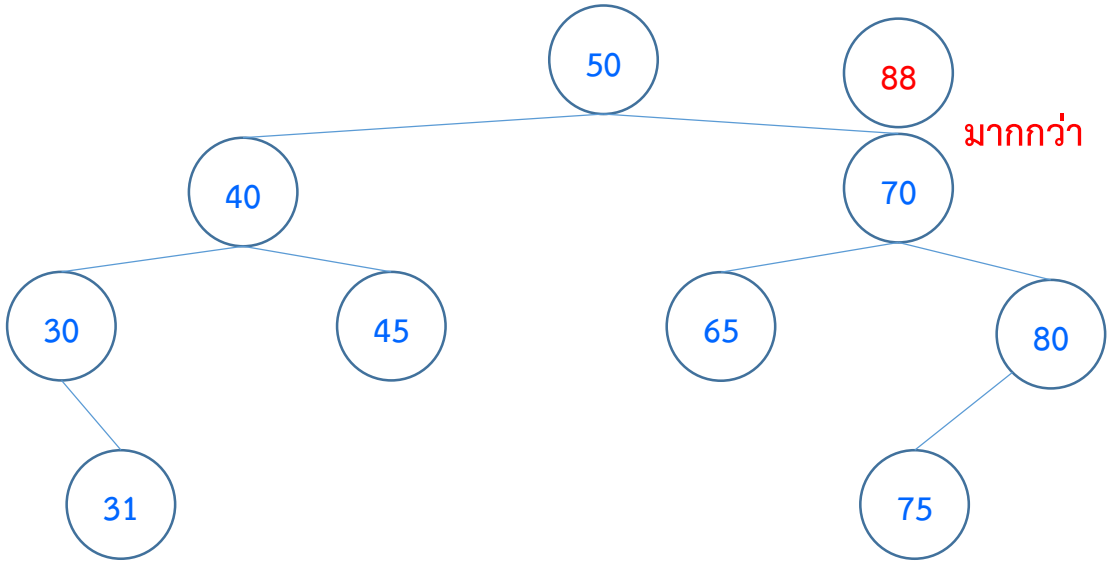
การเพิ่มข้อมูลใน Binary Search Tree

1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
2) หากค่าน้อยกว่า Root ให้ไต่ไปทางซ้าย หากมากกว่าให้ไต่ไปทางขวา
3) ทำซ้ำไปเรื่อย ๆจนกว่าจะถึง node สุดท้าย
4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา



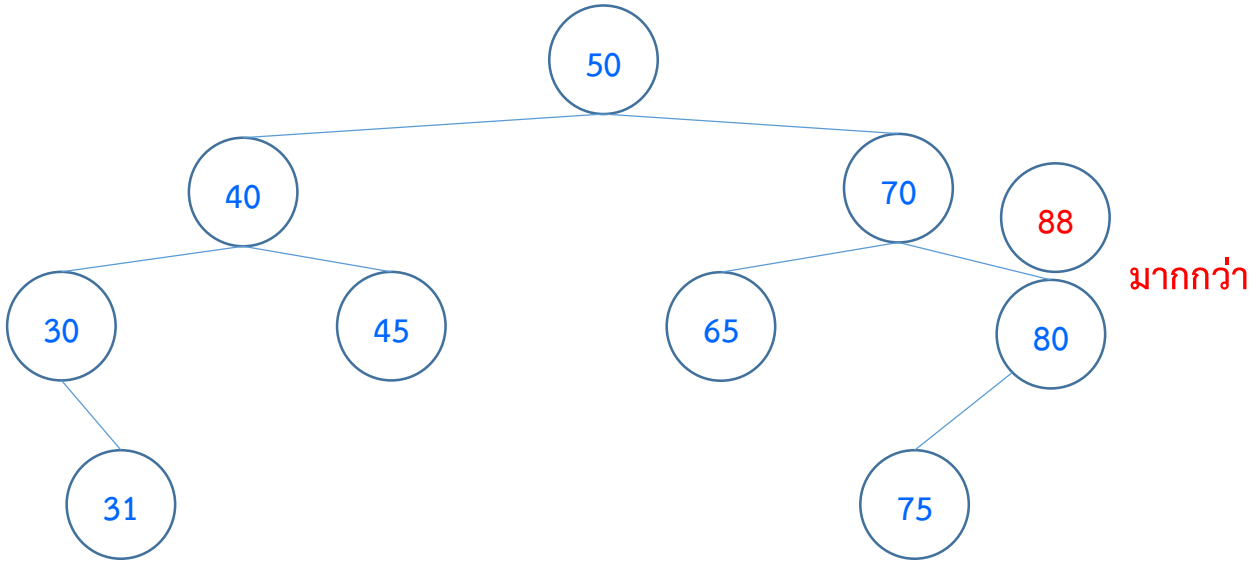
การเพิ่มข้อมูลใน Binary Search Tree

1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
2) หากค่าน้อยกว่า Root ให้ไต่ไปทางซ้าย หากมากกว่าให้ไต่ไปทางขวา
3) ทำซ้ำไปเรื่อย ๆจนกว่าจะถึง node สุดท้าย
4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา



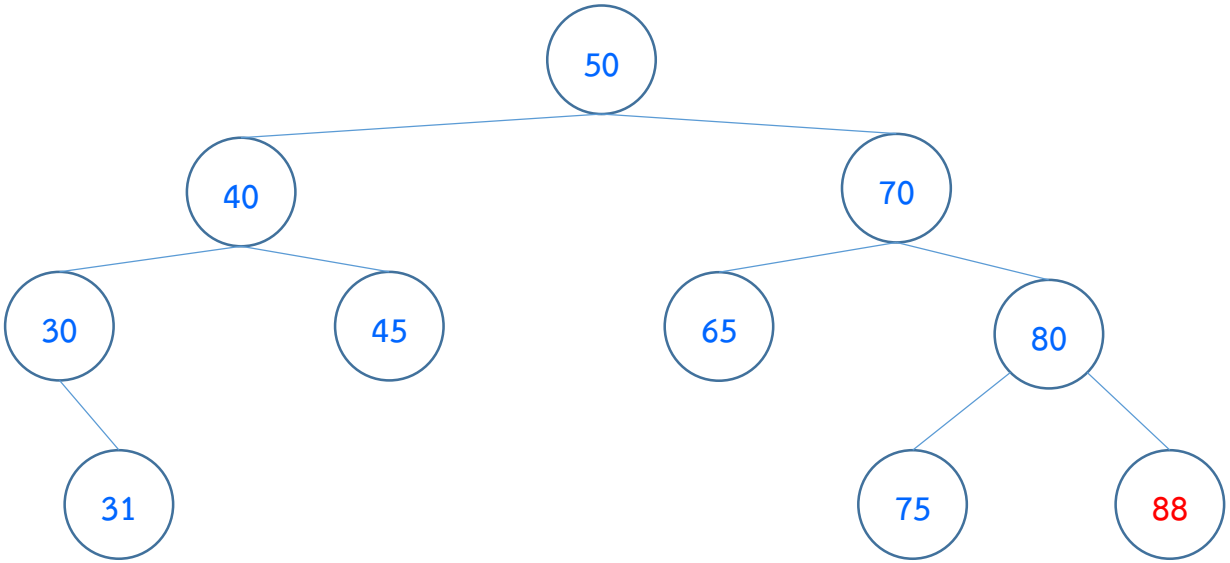
การเพิ่มข้อมูลใน Binary Search Tree

1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
2) หากค่าน้อยกว่า Root ให้ไต่ไปทางซ้าย หากมากกว่าให้ไต่ไปทางขวา
3) ทำซ้ำไปเรื่อย ๆจนกว่าจะถึง node สุดท้าย
4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา



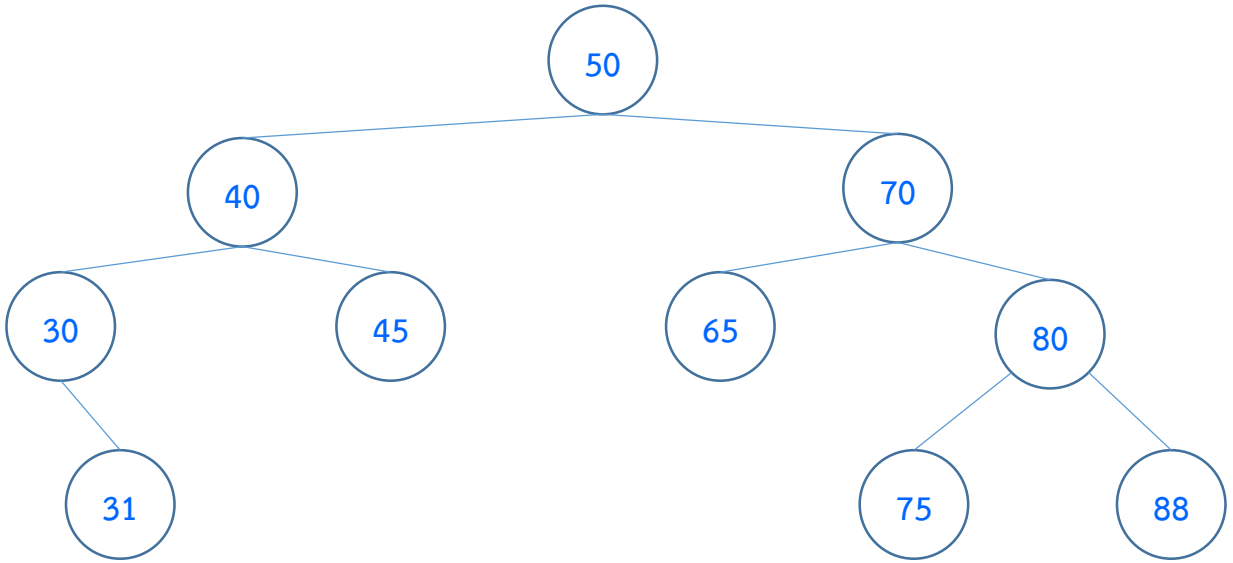
การเพิ่มข้อมูลใน Binary Search Tree

1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
2) หากค่าน้อยกว่า Root ให้ไต่ไปทางซ้าย หากมากกว่าให้ไต่ไปทางขวา
3) ทำซ้ำไปเรื่อย ๆจนกว่าจะถึง node สุดท้าย
4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา



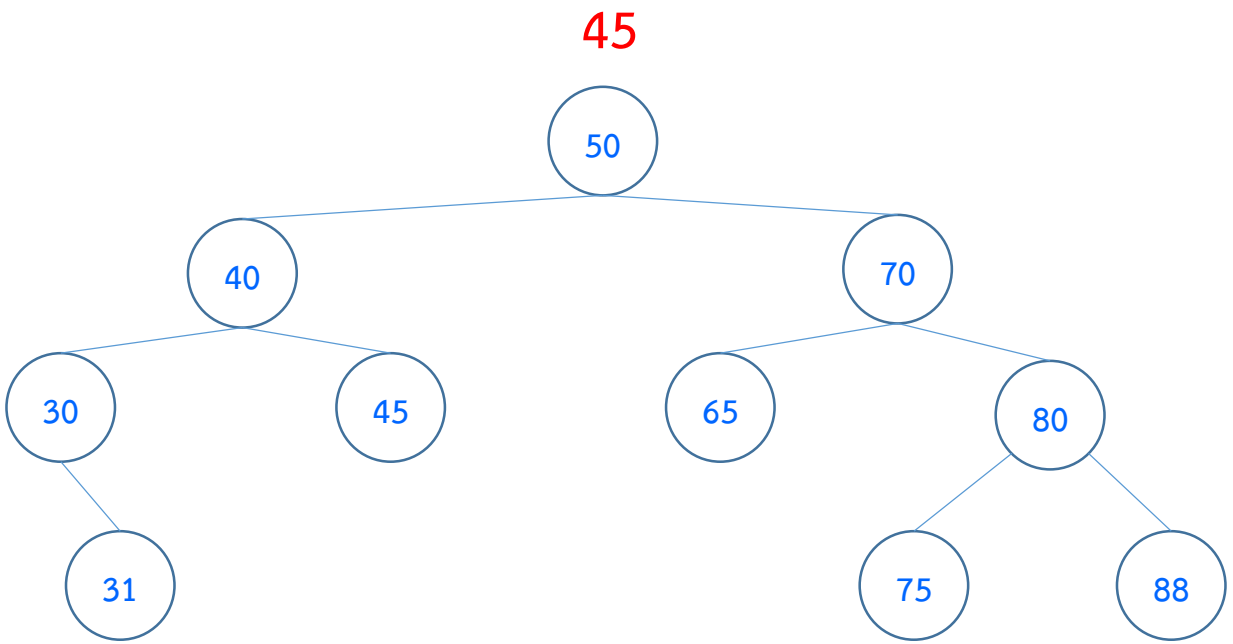
การค้นข้อมูลใน Binary Search Tree

การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



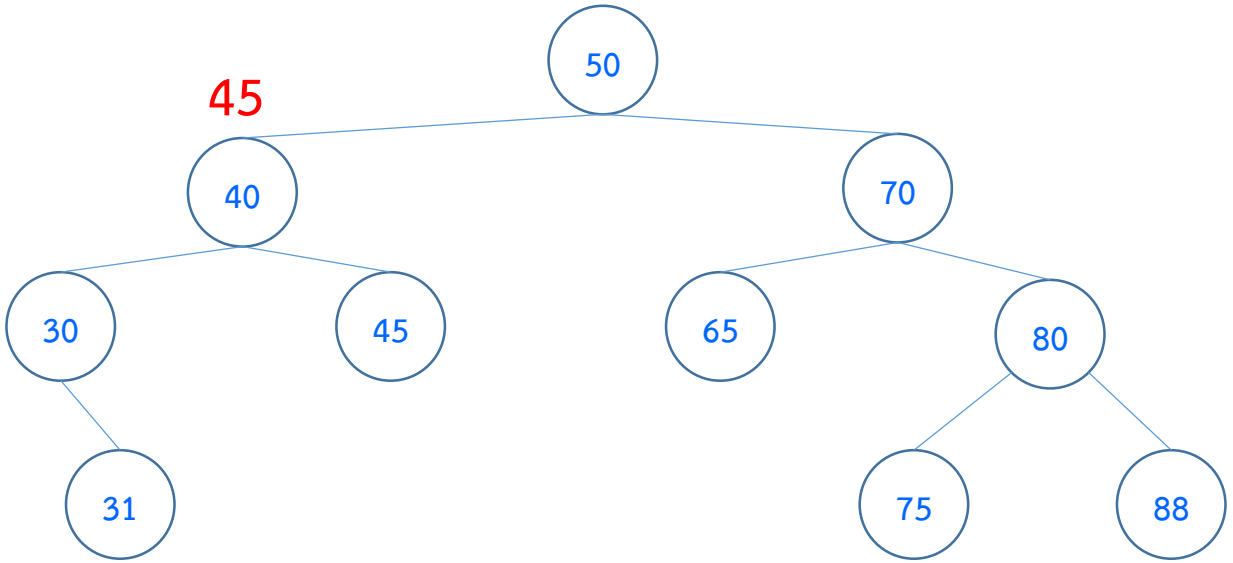
การค้นข้อมูลใน Binary Search Tree

การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



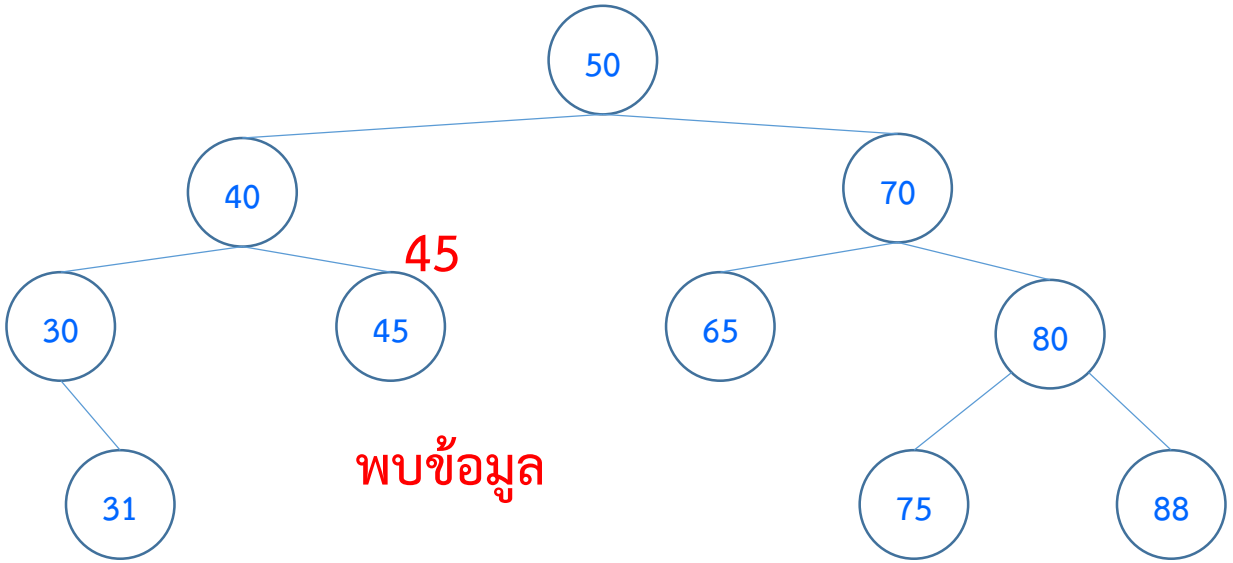
การค้นข้อมูลใน Binary Search Tree

การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



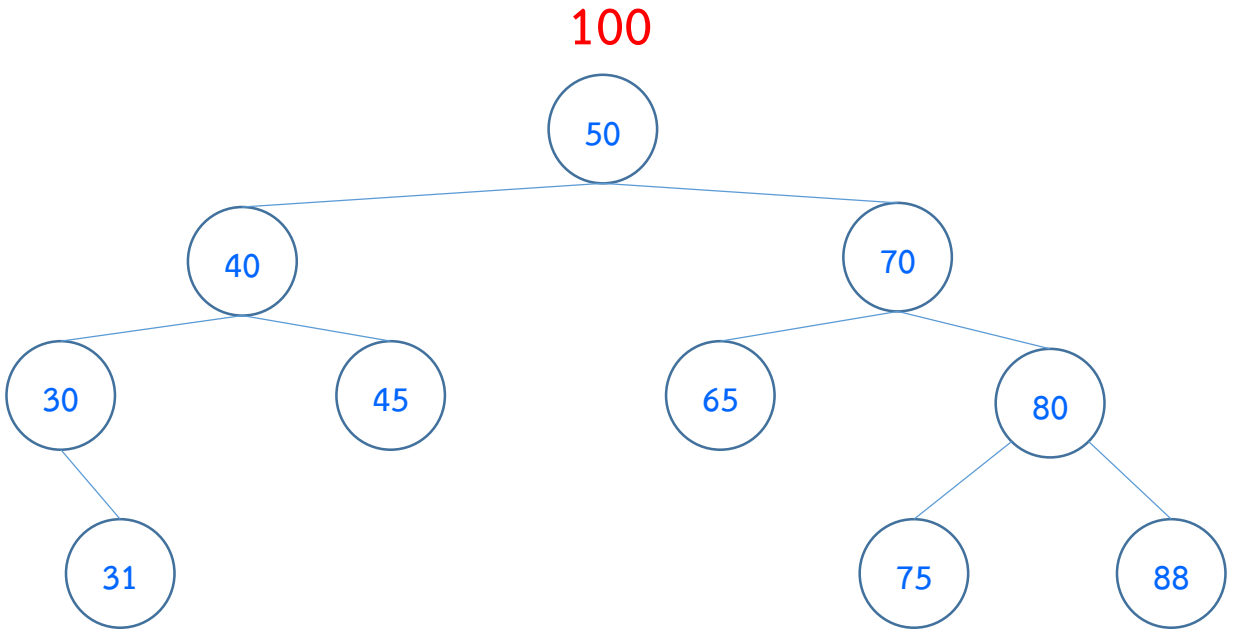
การค้นข้อมูลใน Binary Search Tree

การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



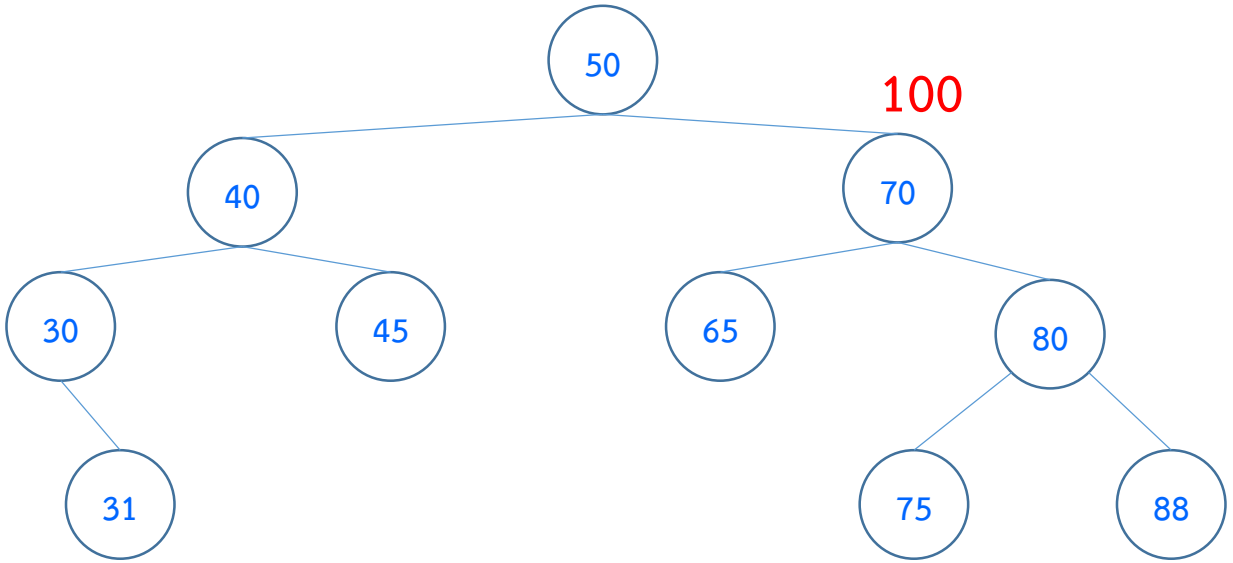
การค้นข้อมูลใน Binary Search Tree

การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



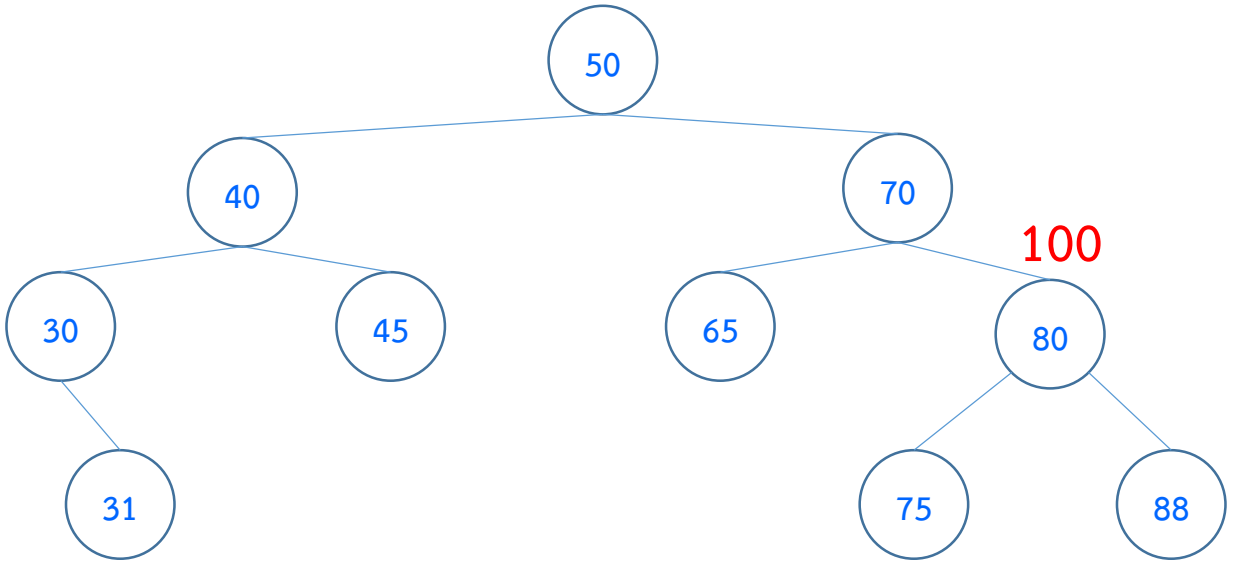
การค้นข้อมูลใน Binary Search Tree

การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



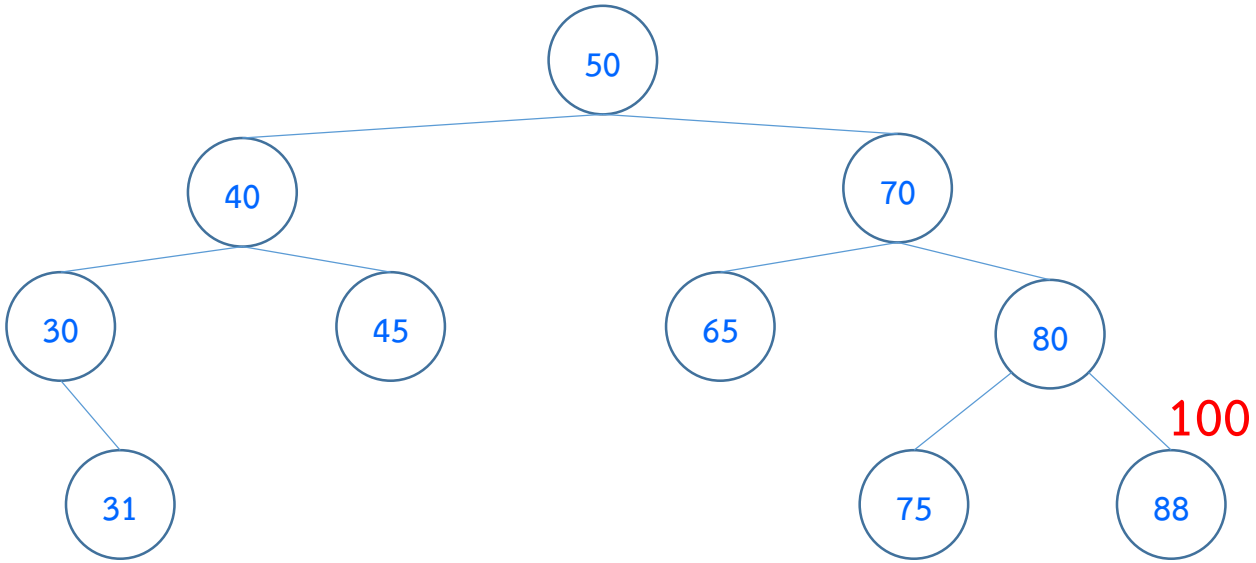
การค้นข้อมูลใน Binary Search Tree

การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



การค้นข้อมูลใน Binary Search Tree

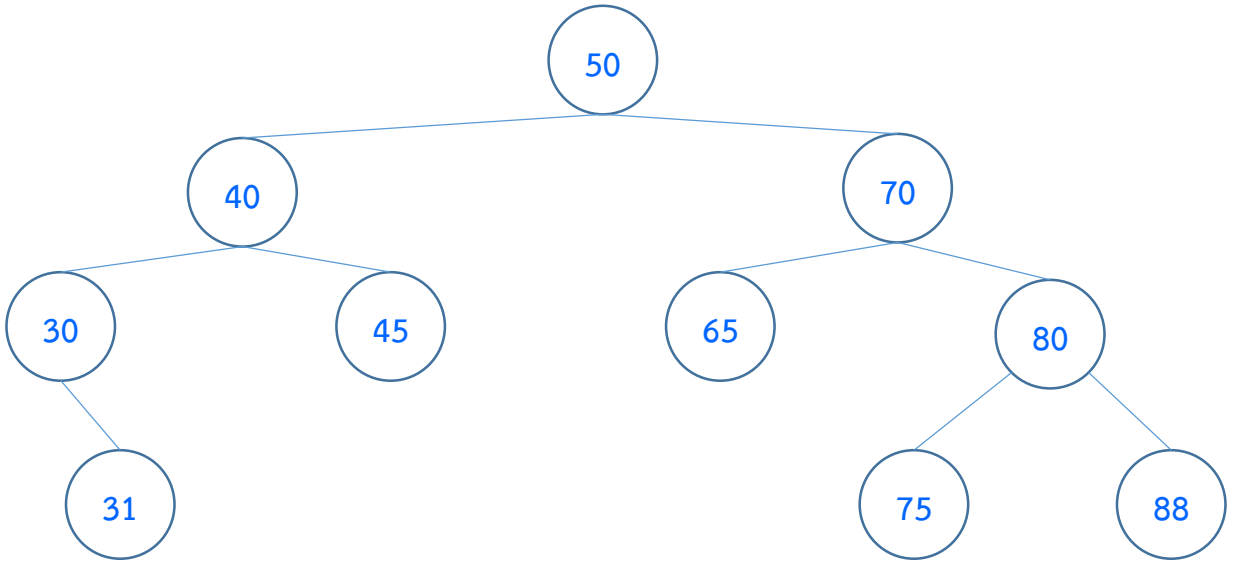
การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



การค้นข้อมูลใน Binary Search Tree

การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf

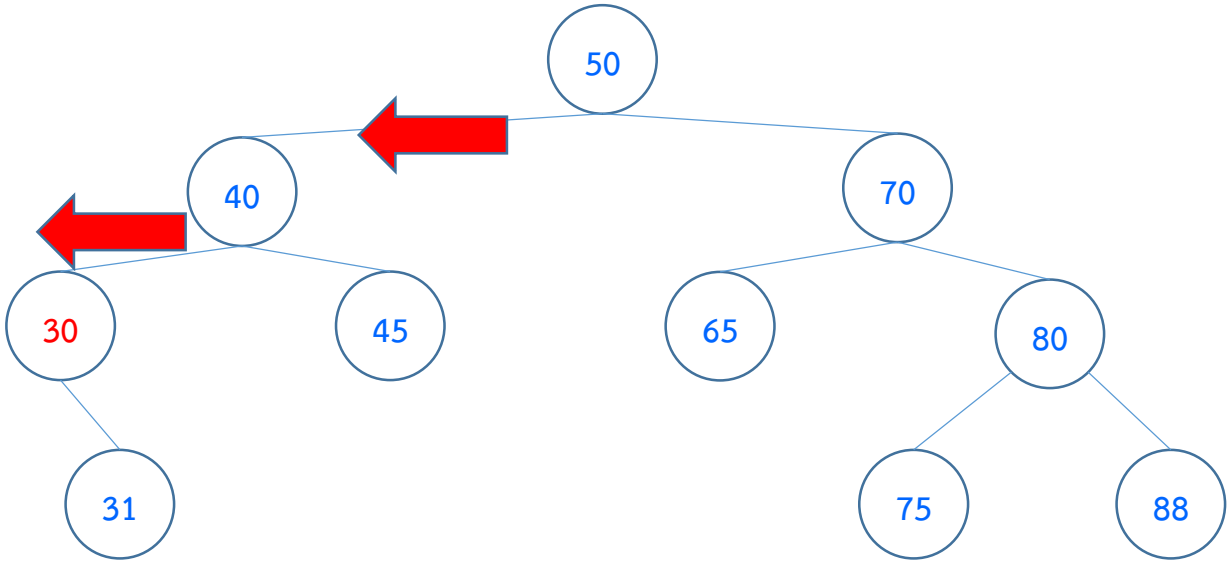
ในกรณีที่ต้นไม้เป็น Complete tree
การค้นข้อมูล ไม่จำเป็นต้องไต่ไปยังทุก node
ทำให้การค้นข้อมูลทำได้เร็วกว่า Array และ Link list



ไม่พบ

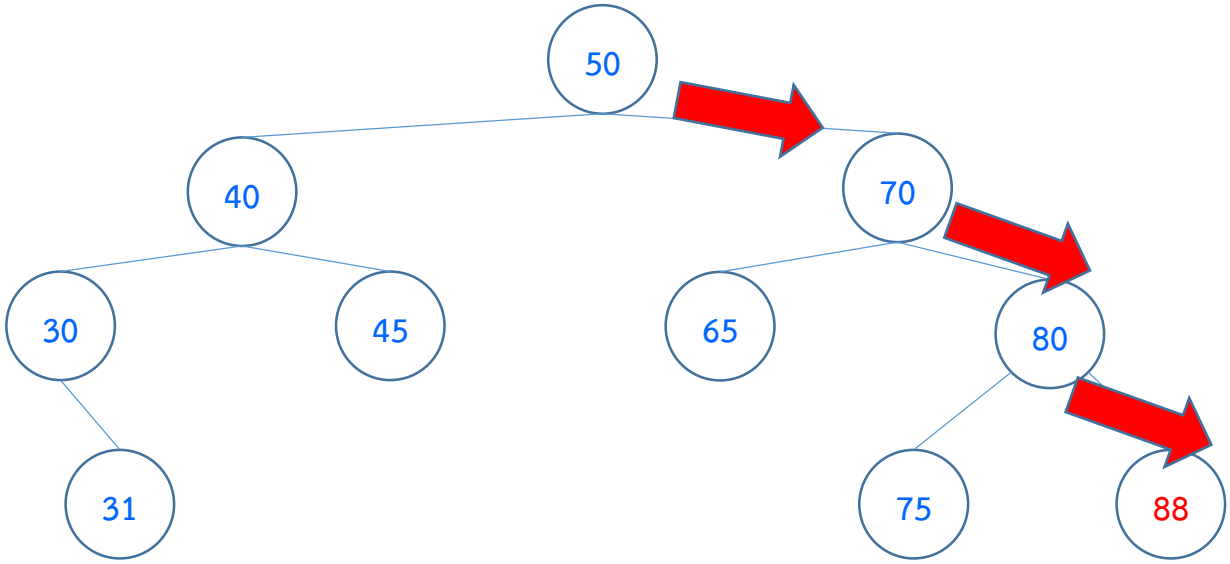
การหาค่าน้อยสุดใน Binary Search Tree

ให้เริ่มจาก Root แล้วไต่มาทางซ้ายตลอด จนกว่าจะไม่มีทางไป node สุดท้ายจะมีค่าน้อยสุดเสมอ



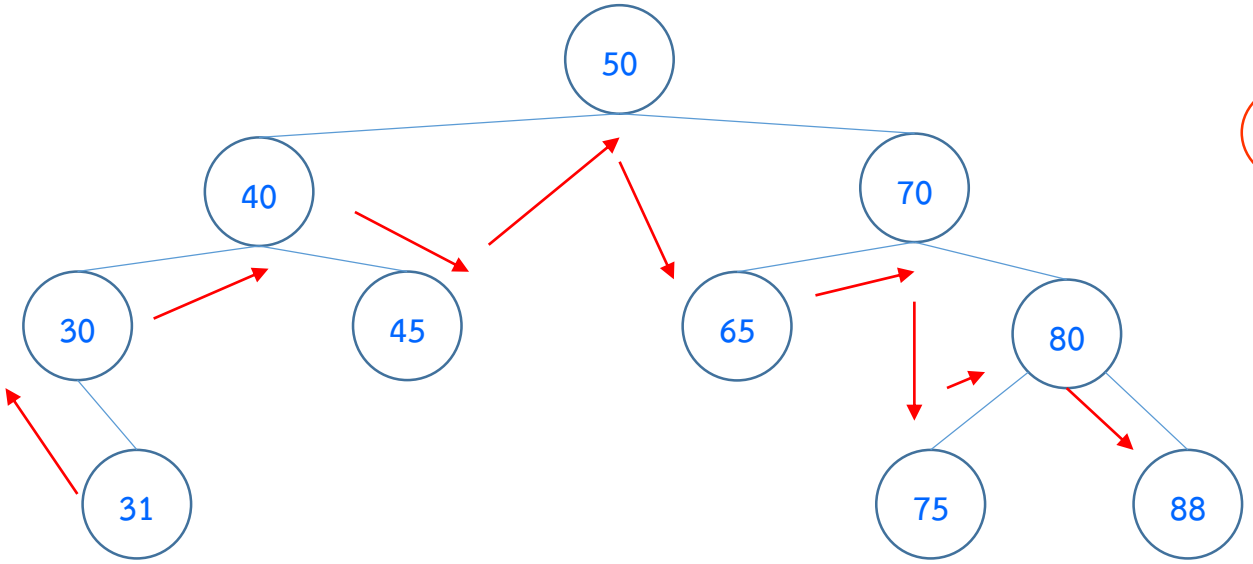
การหาค่ามากสุดใน Binary Search Tree

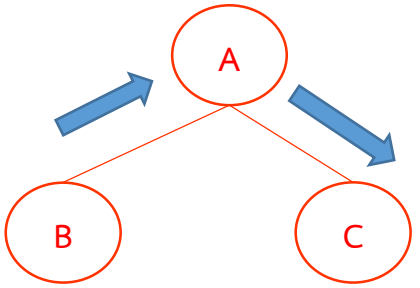
ให้เริ่มจาก Root แล้วไต่มาทางขวาตลอด จนกว่าจะไม่มีทางไป node สุดท้ายจะมีค่าน้อยสุดเสมอ



การเรียงข้อมูลใน Binary Search Tree

ให้ทำการ Traverse แบบ Inorder จะได้ข้อมูลที่เรียงจากน้อยไปมาก เสมอ





การท่องแบบ Inorder

B --> A --> C

31, 30, 40, 45, 50, 65, 70, 75, 80, 88

ความเร็วในการทำงาน

การ Traverse ใช้เวลา O(N)

การ Add ใช้เวลาเท่าความสูงของต้นไม้คือ $O(Log_2N)$ หากต้นไม้ลู่ไปทางเดียว จะใช้เวลา O(N)

การ ค้นข้อมูล ใช้เวลาเท่าความสูงของต้นไม้คือ $O(Log_2N)$ หากต้นไม้ลู่ไปทางเดียว จะใช้เวลา O(N)

ต้นไม้ลู่ไปทางเดียว = Linked-list

ทำไมเราถึงใช้ต้นไม้ในการเก็บข้อมูล ?

ในการใช้งานจริง ต้นไม้มักจะเป็น Complete tree

Complete tree จะสูง Log_2N

หากต้องการค้นข้อมูล 1 ล้าน Record ใน Array หรือ Linked List จะใช้เวลา O(N)
ต้องเปรียบเทียบ 1 ล้านครั้ง

หากใช้ต้นไม้ จะใช้เวลา $O(Log_2N)$

$Log_2(1000000)=19.93$

หรือประมาณ 20 ครั้งเท่านั้น

ค้นหาข้อมูลประชากรทั้ง 69 ล้านคน หากเก็บเป็น Complete tree
จะต้องวน loop ในการค้นกี่ครั้ง

$Log_2(69000000)=26.04$
หรือ 27 ครั้งเท่านั้น

ความเร็วในการค้น แลกมาด้วยอะไร ?

Linked list ใช้เวลาในการเพิ่มข้อมูล O(1)
แต่ ต้นไม้ใช้เวลา $Log_2(N)$

จึงเป็นการแลกที่คุ้มค่า

สอบ Final

28/10/2562 8.00-11.00

80-403 อาคารเรียนรวม 80 ปี

จดเข้าได้ 1 แผ่น A4 และใช้เครื่องคิดเลขได้

ขอให้ทุกคนโชคดี

เจอกันใหม่ ปี 3