

CS221 Data Structure

Chapter 7

Trees



By Dr. Paween Khoenkaw
Computer Science MJU

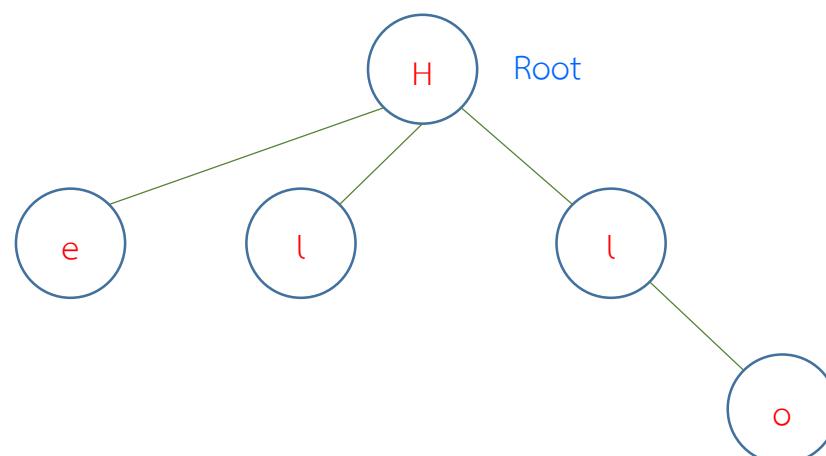


Trees:

Tree หรือ ต้นไม้ เป็น แบบชนิดข้อมูลนามธรรม ประเภทหนึ่ง มีลักษณะการเรียงเป็นกิ่งก้านสาขา แตกแขนงออกไป จะไม่มีวงวน (loop) โดยในสมาชิกตัวต่าง ๆ โดยสมาชิกจะถูกเก็บไว้ในประเภท ข้อมูลชนิดวัตถุ (Object) หรือโครงสร้าง (Structure) เรียกว่าปม (node) ซึ่งจะมีตัวแปรซึ่งเก็บตัว ชี้ (Pointer) ไปยังปมอื่น ๆ ได้

ส่วนประกอบของต้นไม้จะประกอบด้วย

- 1) Node หรือปม คือสมาชิกของต้นไม้ใช้สำหรับเก็บข้อมูล แทนด้วยวงกลม
- 2) Edge หรือ กิ่ง ใช้ในการเชื่อม node บางครั้งเรียกว่า path แทนด้วยเส้น
- 3) ต้นไม้จะโตจำกัดบนลงไปด้านล่าง node ที่อยู่บนสุดจะเรียกว่า Root หรือราก



Trees:

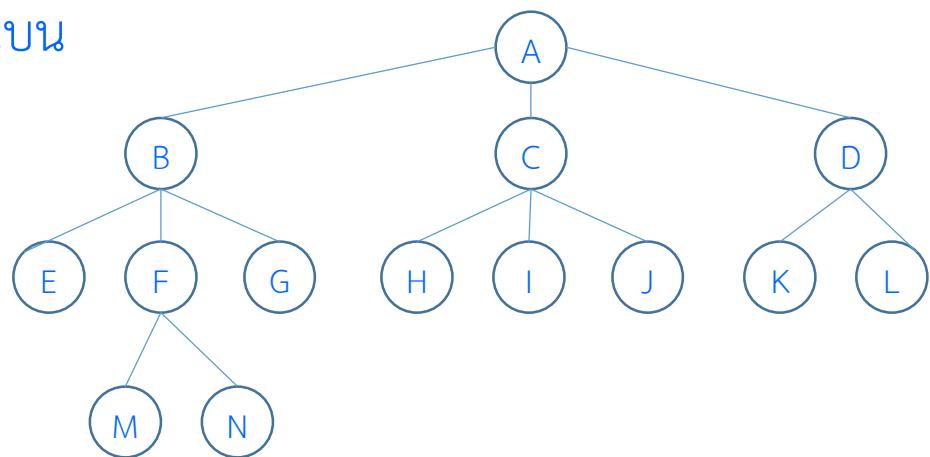
โครงสร้างข้อมูลแบบต้นไม้ นั้นมีต้นกำเนิดมาจากการนิศาสตร์ในสาขาทฤษฎีกราฟ เพื่อให้เข้าใจ ตรงกัน จึงนามคุณลักษณะและคุณสมบัติของต้นไม้ดังต่อไปนี้

Parent node หรือปัมพ่อ ใช้เรียก Node ที่อยู่ด้านบน

ตัวอย่างเช่น

B เป็น Parent ของ E,F,G

F เป็น Parent ของ M,N



Trees:

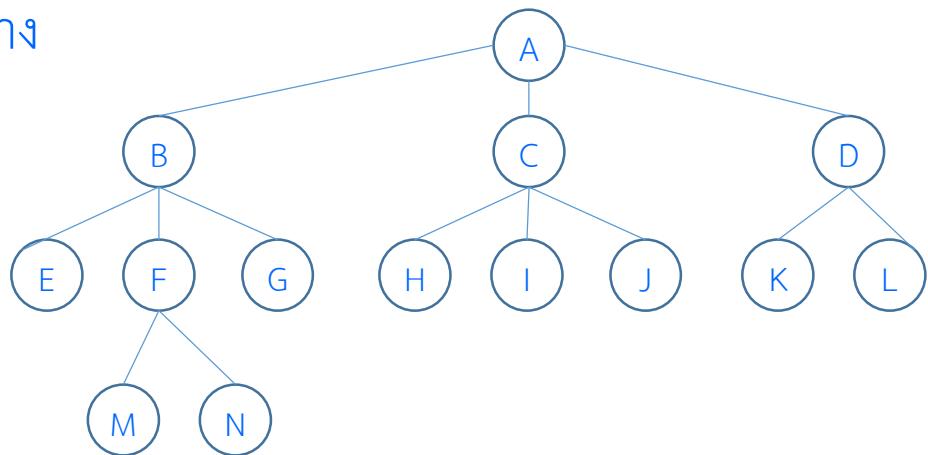
โครงสร้างข้อมูลแบบต้นไม้ นั้นมีต้นกำเนิดมาจากการนิศาสตร์ในสาขาทฤษฎีกราฟ เพื่อให้เข้าใจ ตรงกัน จึงนามคุณลักษณะและคุณสมบัติของต้นไม้ดังต่อไปนี้

Child node หรือปมลูก ใช้เรียก Node ที่อยู่ด้านล่าง

ตัวอย่างเช่น

D เป็น Child node ของ A

K,L เป็น Child node ของ D



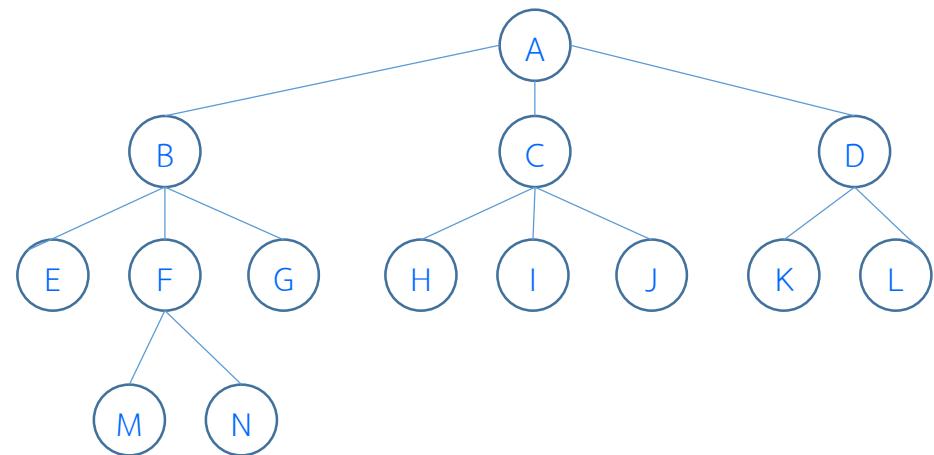
Trees:

Siblings หรือปมพี่น้อง ใช้เรียก Node ที่มีปมพ่อเดียวกัน

ตัวอย่างเช่น

E,F,G เป็น Siblings

M, N เป็น Siblings



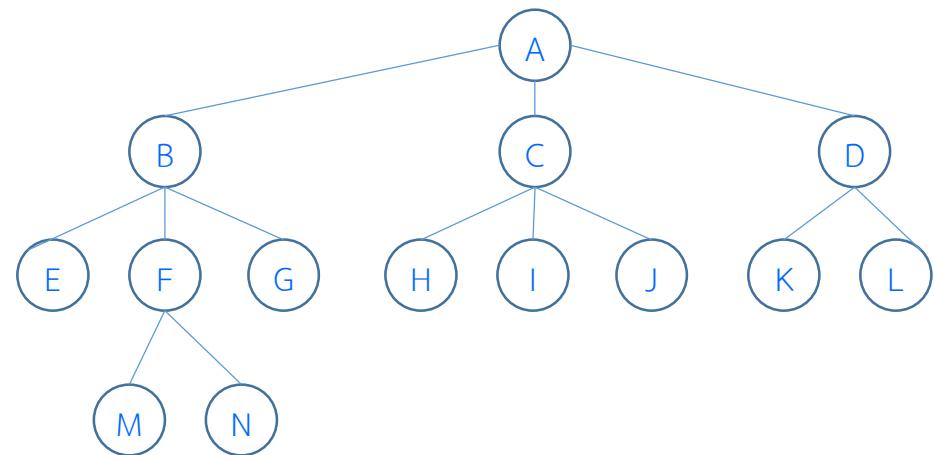
หาก Siblings มีการเรียงลำดับ ในรูปแบบใดรูปแบบหนึ่ง จะเรียกต้นไม้นี้ว่า ordered tree

Trees:

Leaf node หรือปมใบ คือปมที่ไม่มีปมลูก

ตัวอย่างเช่น

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



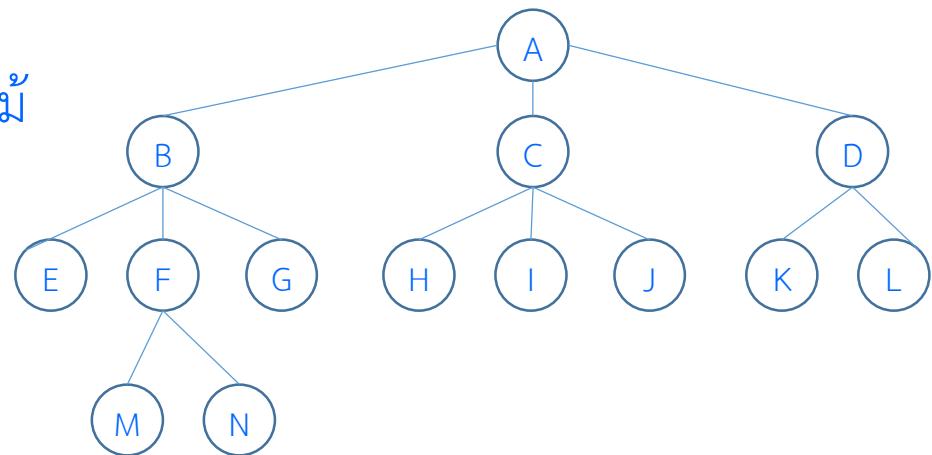
Trees:

เส้นทางจาก Root มายัง node ใด ๆ จะมีได้เพียงเส้นทางเดียวเท่านั้น

การเดินทางมายัง N จะมีได้เส้นทางเดียวเท่านั้นคือ

A, B, F, N

หากมีมากกว่า 1 เส้นทาง โครงสร้างนี้จะไม่ใช่ต้นไม้



Node ที่อยู่ระหว่างทาง ย้อนขึ้นไปจนถึง Root จะเรียกว่า Ancestors

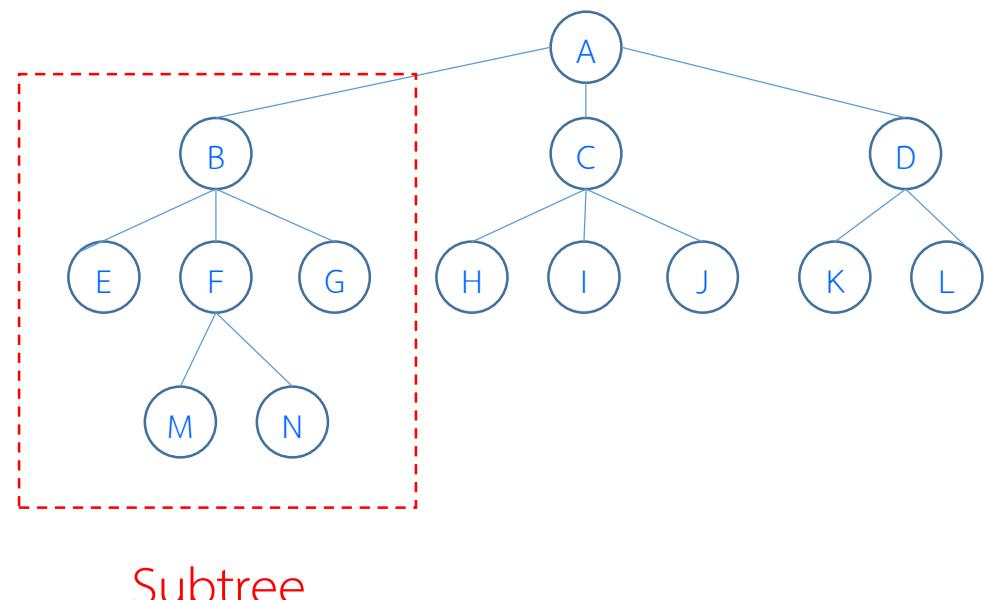
ตัวอย่างเช่น Ancestors ของ N คือ A, B, F

Trees:

Descendent คือ Node ทั้งหมดที่อยู่ด้านล่าง

ตัวอย่างเช่น Descendants ของ B คือ
E,F,G,M,N

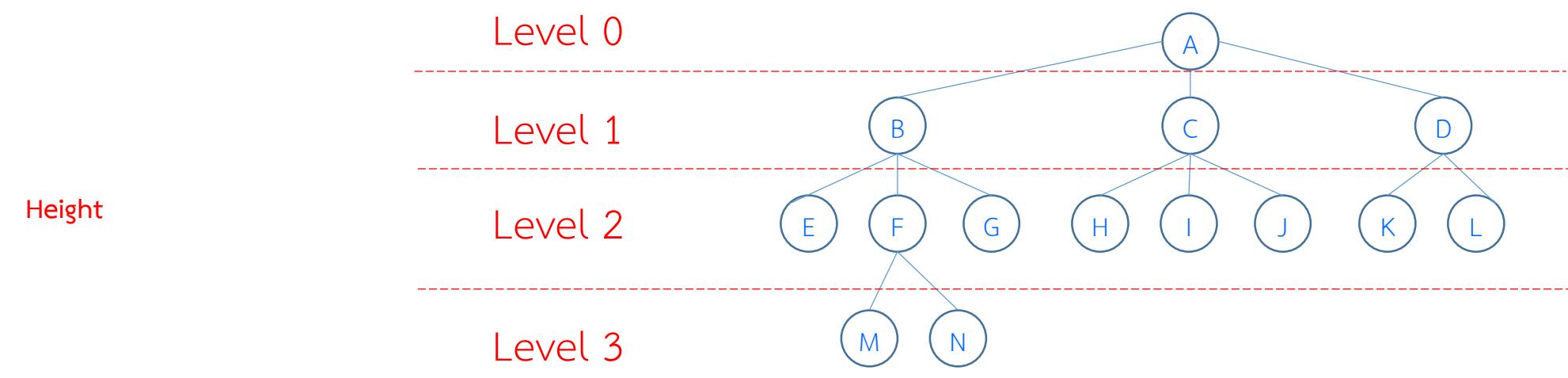
Descendants ยังคงโครงสร้างต้นไม้ไว้
จึงเรียกได้ว่าเป็น Subtree หรือ ต้นไม้มีอยู่
โดยมี B เป็น Root



Trees:

ระดับ หรือ ความลึก ของต้นไม้

การนับระดับของต้นไม้ จะเริ่มนับจาก Root เป็นระดับ 0
และเพิ่มระดับขึ้นเรื่อยๆ ในแต่ละชั้น



จำนวนระดับสูงสุดของต้นไม้ คือความสูงของต้นไม้

ตัวอย่างเช่นต้นไม้ต้นนี้มีความสูงเป็น 3

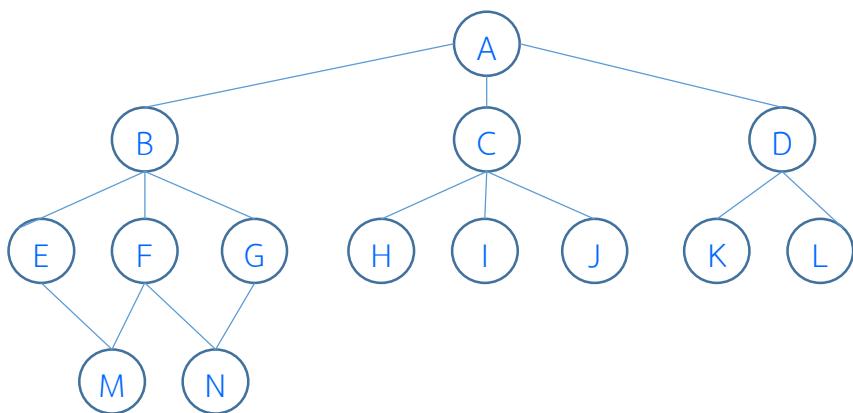
ต้นไม้ที่มีเพียง Root node จะมีความสูงเป็น 0

ต้นไม้ว่าง จะมีความสูงเป็น -1 หรือ NULL

Trees:

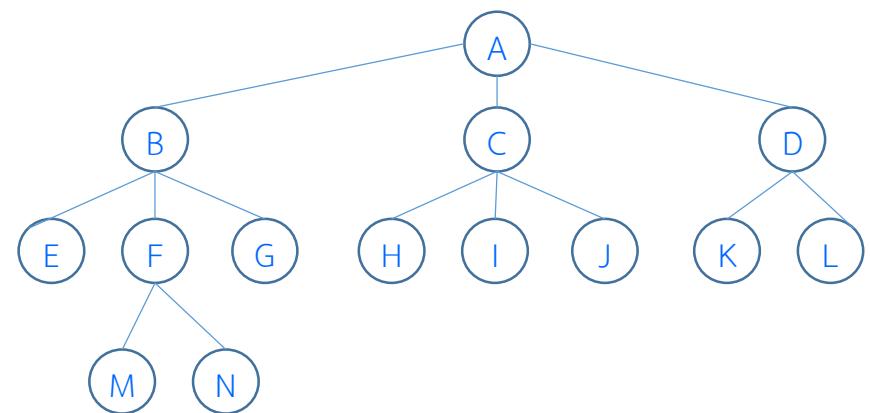
คุณสมบัติที่สำคัญของต้นไม้

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



ไม่ใช่ต้นไม้

โครงสร้างนี้เรียกว่ากราฟ

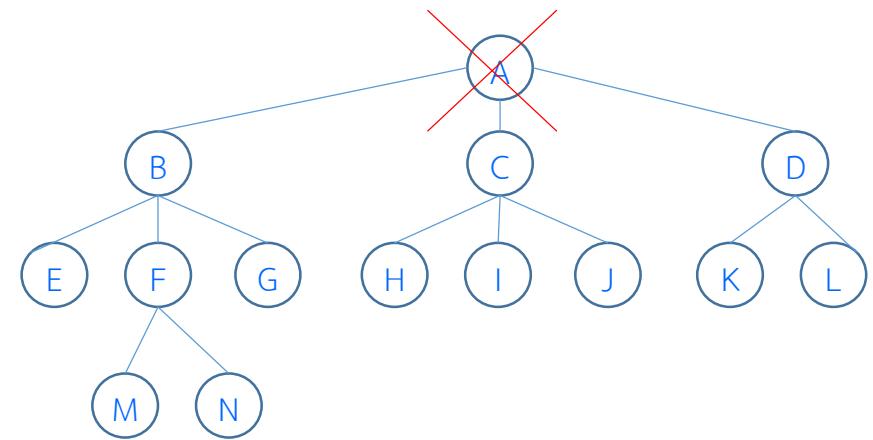


ต้นไม้

Trees:

คุณสมบัติที่สำคัญของต้นไม้

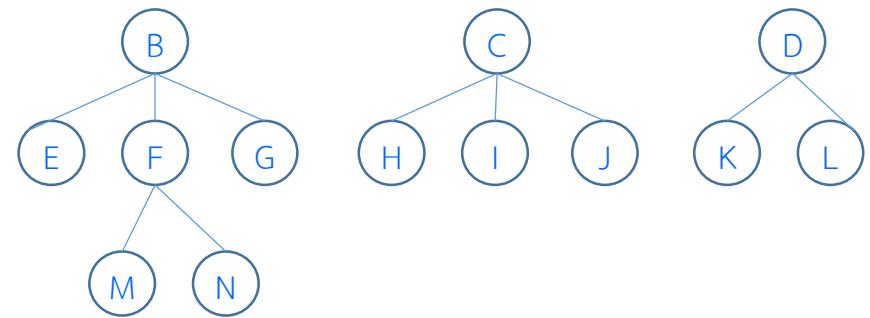
- โครงสร้างไม่เป็นเชิงเส้น
- เส้นทางจาก Root ไปยัง Node ใด ๆ มีเพียงเส้นทางเดียว
- ทุก node มี parent เดียว ยกเว้น root node



Trees:

คุณสมบัติที่สำคัญของต้นไม้

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



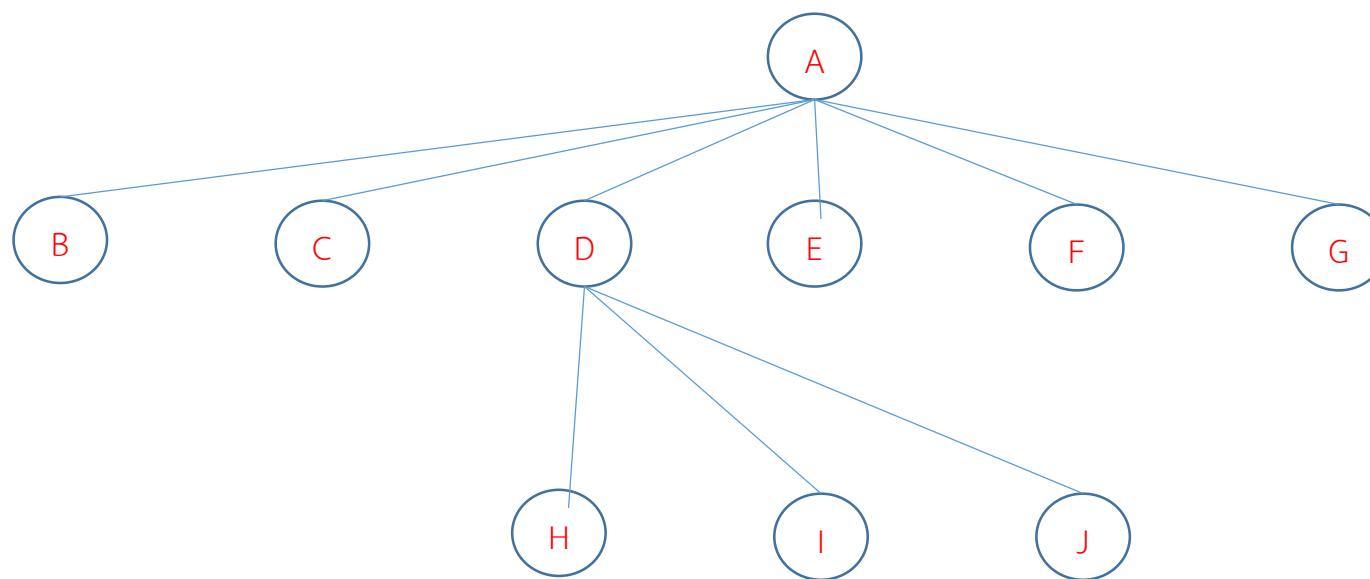
Forest

Trees:

การจำแนกประเภทต้นไม้

ต้นไม้นั้นจะจำแนกประเภทจากจำนวน ลูก สูงสุดที่แตกต่างกัน node จะมีได้

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



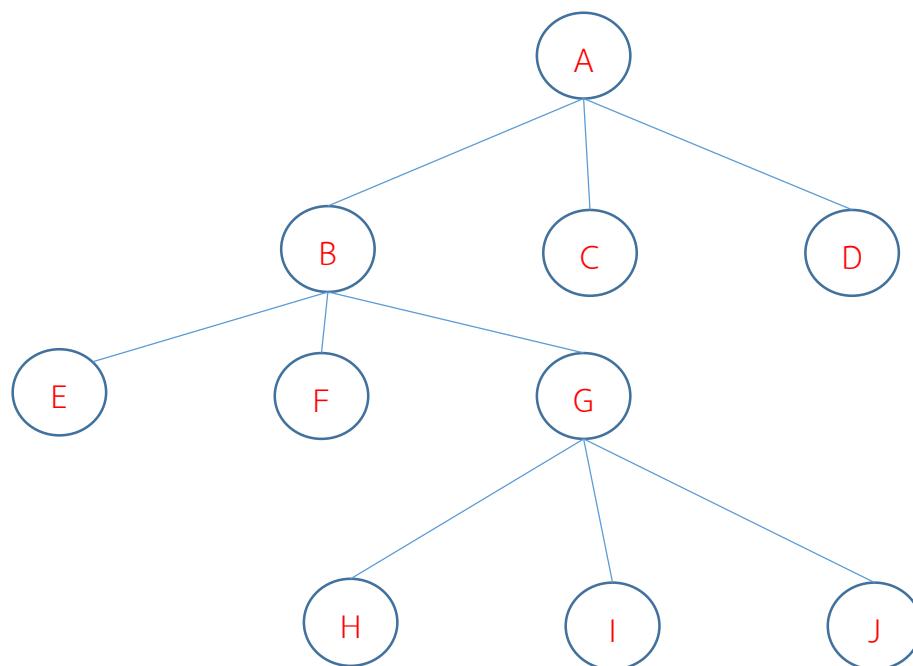
หาก node มีรูปแบบลำดับการเรียง จะเรียกว่า Multiway - tree

Trees:

การจำแนกประเภทต้นไม้

ต้นไม้นั้นจะจำแนกประเภทจากจำนวน ลูก สูงสุดที่แต่ละ node จะมีได้

ต้นไม้ที่จำนวนลูกสูงสุด 3 เรียกว่า ternary tree

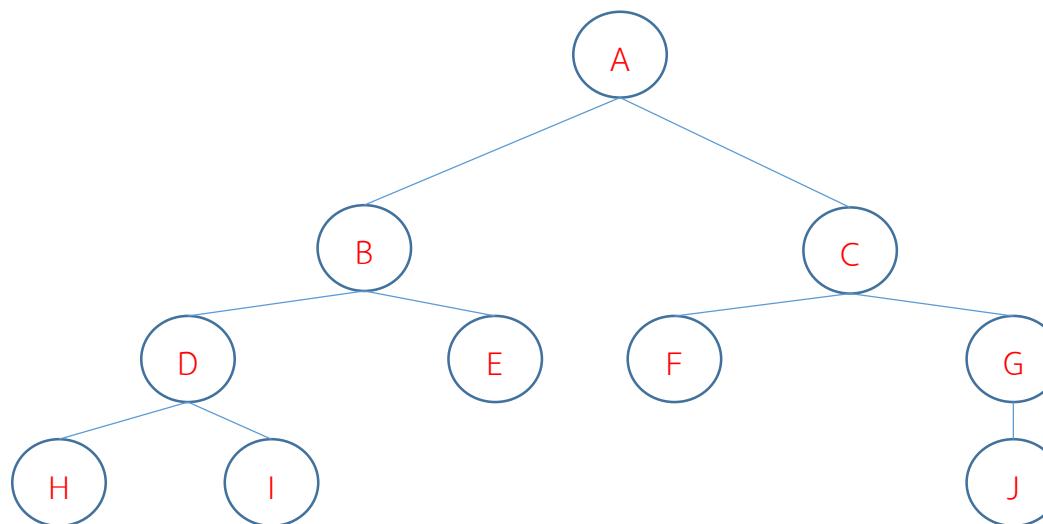


Trees:

การจำแนกประเภทต้นไม้

ต้นไม้นั้นจะจำแนกประเภทจากจำนวน ลูก สูงสุดที่แตกต่างกัน node จะมีได้

ต้นไม้ที่จำนวนลูกสูงสุด 2 เรียกว่า Binary tree หรือต้นไม้ทวิภาค



ต้นไม้ใช้ทำอะไรได้บ้าง

ฝังองค์กร



Logo of Faculty of Science, Maejo University, featuring a circular emblem with a tree and the text "Faculty of Science" and "มหาวิทยาลัยแม่โจ้".

คณะวิทยาศาสตร์ มหาวิทยาลัยแม่โจ้
FACULTY OF SCIENCE, MAEJO UNIVERSITY

.. หน้าหลัก .. เกี่ยวกับคณะวิทยาศาสตร์ .. หน่วยงาน .. การศึกษา .. บุคลากร .. รอบรั้วมหาวิทยาลัย



A photograph of the entrance to the Faculty of Science building at Maejo University. The building has a modern design with large glass windows and doors. A blue sign above the entrance reads "อาคารจุฬารัตน์" (Building J) and "เกี่ยวกับคณะวิทยาศาสตร์" (About the Faculty of Science). The sky is clear and blue.

โครงสร้างการบริหาร

Organizational chart of the Faculty of Science:

```
graph TD; Root[คณบดีคณะวิทยาศาสตร์] --- Admin[คณกรรวมประจำคณะ]; Root --- Dept[รองคณบดี]; Root --- Prof[เลขาธุการคณะ]; Dept --- Dept1[รองคณบดีฝ่ายวิชาการและบริการวิชาการ]; Dept1 --- Dept1_1[รองคณบดีฝ่ายวิชาการและวิเทศสัมพันธ์]; Dept1 --- Dept1_2[รองคณบดีฝ่ายบริหารและพัฒนาทรัพยากรบุคคล]; Dept1 --- Dept1_3[รองคณบดีฝ่ายกิจการนักศึกษาและกิจกรรมพิเศษ]; Dept1 --- Dept1_4[รองคณบดีฝ่ายยุทธศาสตร์]; Dept1 --- Dept1_5[และผู้ทรงคุณวุฒิ]; Dept1_1 --- Admin1[งานบริหารและธุรการ]; Dept1_2 --- Admin2[งานคลังและพัสดุ]; Dept1_3 --- Admin3[งานบริการการศึกษาและกิจกรรมนักศึกษา]; Dept1_4 --- Admin4[งานนโยบาย แผน และประเมินคุณภาพ]; Dept1_5 --- Admin5[งานบริการวิชาการและวิจัย]; Dept1_1 --- Prof1[งานบริหารและธุรการ]; Dept1_2 --- Prof2[งานคลังและพัสดุ]; Dept1_3 --- Prof3[งานบริการการศึกษาและกิจกรรมนักศึกษา]; Dept1_4 --- Prof4[งานนโยบาย แผน และประเมินคุณภาพ]; Dept1_5 --- Prof5[งานบริการวิชาการและวิจัย]; Admin --- Admin1; Admin --- Admin2; Admin --- Admin3; Admin --- Admin4; Admin --- Admin5; Prof --- Prof1; Prof --- Prof2; Prof --- Prof3; Prof --- Prof4; Prof --- Prof5;
```

16

Trees:

ต้นไม้ใช้ทำอะไรได้บ้าง

โครงสร้างเพิ่มข้อมูล

โครงสร้างไฟล์ในเครื่องคอมพิวเตอร์

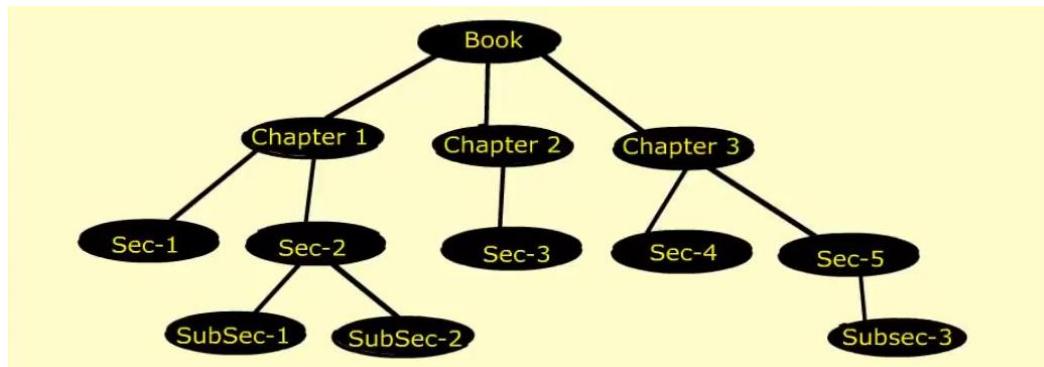
Name	Date modified	Type
manifest.json	2/2/2018 11:08 AM	JSON File
WCChromeNativeMessagingHost.exe	2/2/2018 11:08 AM	Application

```
graph TD; /C --- Dir1[Dir 1]; /C --- Dir2[Dir 2]; /C --- Dir3[Dir 3]; Dir1 --- File1[File 1]; Dir1 --- File2[File 2]; Dir2 --- File4[File 4]; Dir2 --- File3[File 3]; Dir3 --- File7_1[File 7]; Dir3 --- File6_1[File 6]; Dir3 --- File7_2[File 7]; Dir3 --- SubDir3[Sub dir-3]; SubDir3 --- File6_2[File 6]; SubDir3 --- File5[File 5];
```

Trees:

ต้นไม้ใช้ทำอะไรได้บ้าง

โครงสร้างเอกสาร



SoftwareBasedMethodForUltrasonicRangeFinderAccuracyImprovement...

File Home Insert Design Layout Reference Mailings Review View EndNote Foxit Read Tell me... Sign in Share

Times New Roman AaBbCcDd AaBbCcDd AaBbCcDd Editing

Normal No Spac... Heading 6

Font Paragraph Styles

size of method.

3	2000 ±0	1.940 ±2.406	1.249 ±2.004	2.070 ±2.505	1.358 ±2.003
4	1 ±0	1.079 ±0.527	0.909 ±0.757	1.209 ±0.557	0.872 ±0.614
5	115.79 ±83.987	1.360 ±0.454	0.962 ±0.472	1.490 ±0.548	0.972 ±0.558

This research was partially supported by National Science Technology and Innovation Policy Office. The authors would like to thank the Northern Talent Mobility Clearing House and The colleagues at INTNN Laboratory, Faculty of Science, Maejo University for their support.

The accuracy of method 2 and 3 is deepened on the size of the window. In these experiments the relationship between window size and measurement error also investigated, and the result is shown in Fig. 6. The trend is shown that the larger window size created more accurate measurement.

Fig. 6. The relationship of measurement error and window size of method.

Front (cm)

Window size

Window size	Front (cm)
0	6.5
500	1.5
1000	1.0
1500	0.8
2000	0.7

SoftwareBasedMethodForUltrasonicRangeFinderAccuracyImprovement...

File Outli Hom Insert Desig Layo Refer Maili Revic View Endt Foxit Tell me... Sign in Share

Level 2 Show Level All Levels

Show Text Formatting Show First Line Only

Master Document Close Outline View Close

size of method.

+ V. CONCLUSION

In this research, the algorithm for ultrasonic range sensor accuracy improvement is proposed. The raw data from four ultrasonic sensors are prerecorded and then process later. The results are compared with four traditional methods. The results are concluded that, using one sample to determine the distance is not accurate and must be avoided, average multiple sample created better result, but it also increased the measurement time, the proposed algorithm created the same result with the average method, but required significantly less sample, this results in the faster measurement time, the best method is to use multiple sensors, but this method also increases the hardware cost and also increase the size of circuit footprint. The temperature compensation also required to produce the accurate measurement. However, using the estimation method or precise method to calculate the speed of sound do not affect the result in overall results.

+ REFERENCES

[1] A. Carullo and M. Parvis, "An ultrasonic sensor for distance measurement in automotive applications," *IEEE Sensors Journal*, vol. 1, p. 143, 2001.

[2] O. Intharasombat and P. Khoenkaw, "A low-cost flash flood monitoring system," in *2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2015, pp. 476-479.

[3] S. Flores, J. Geil, and M. Vossiek, "An ultrasonic sensor network for high-quality range-bearing-based indoor positioning," in *2016 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, 2016, pp. 572-576.

[4] (11/12/2016). Ultrasonic Ranging Module HC - SR04 Available:
<http://www.micropik.com/PDF/HCSR04.pdf>

[5] J. Majchrzak, M. Michalski, and G. Wiczynski, "Distance Estimation With a Long-Range Ultrasonic Sensor System," *IEEE Sensors Journal*, vol. 9, pp. 767-773, 2009.

[6] D. P. R. K. K. Anitha, V. Vamsi Sudheera, M. Narendra Kumar, "Time-of-Flight Measurement for Ultrasonic Sensors using Digital Signal Processing Techniques," *International Journal of electronics & communication technology*, vol. 2, p. 267, 2011.

[7] A. B. a. R. S. L. M. Leopoldo Angrisani, "Ultrasonic

+ ACKNOWLEDGMENT

This research was partially supported by National Science Technology and Innovation Policy Office. The authors would like to thank the Northern Talent Mobility Clearing House and The colleagues at INTNN Laboratory, Faculty of Science, Maejo University for their support.

+ REFERENCES

[1] [1] A. Carullo and M. Parvis, "An ultrasonic sensor for distance measurement in automotive applications," *IEEE Sensors Journal*, vol. 1, p. 143, 2001.

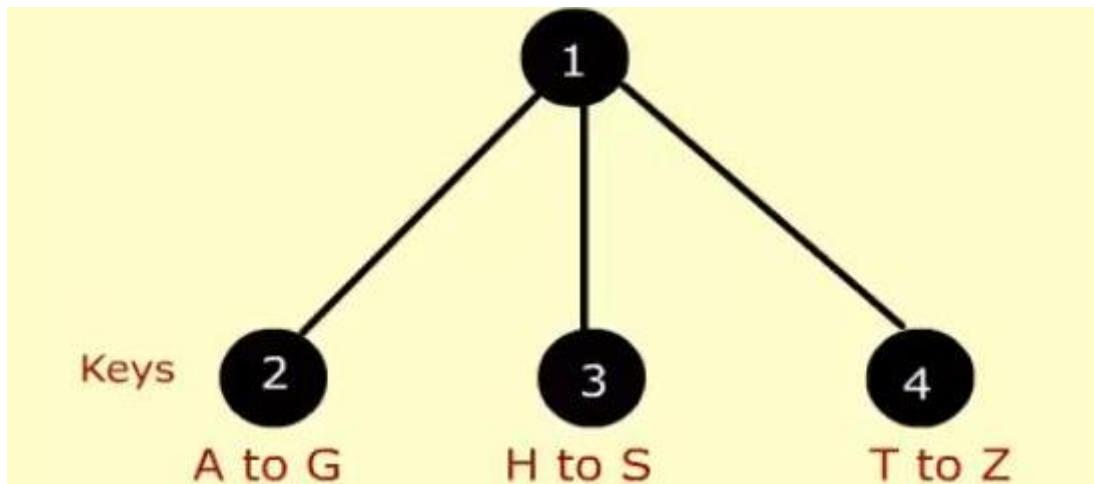
[2] [2] O. Intharasombat and P. Khoenkaw, "A low-cost flash flood monitoring system," in *2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2015, pp. 476-479.

[2] [2] S. Flores T. Geil and M. Vossiek, "An ultrasonic

Trees:

ต้นไม้ใช้ทำอะไรได้บ้าง

ฐานข้อมูล



Milti-way tree ใช้ในการเก็บข้อมูลในระบบฐานข้อมูล เพื่อให้สืบค้นได้เร็ว

Trees:

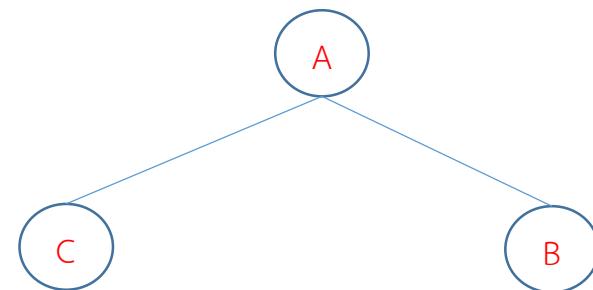
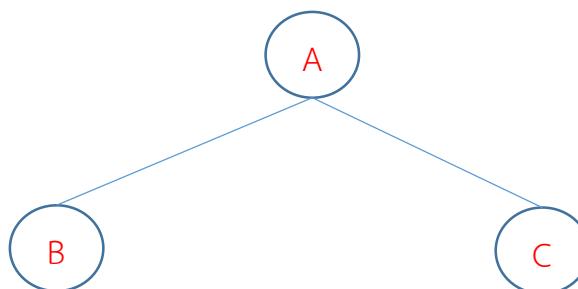
วิชานี้เราจะศึกษาต้นไม้ที่วิภาคเป็นหลัก

We will focus on Binary Tree fist!



ต้นไม้ทวิภาค

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



ต้นไม้ 2 ต้นนี้มีข้อมูลเมื่อนกัน แต่มีโครงสร้างต่างกัน

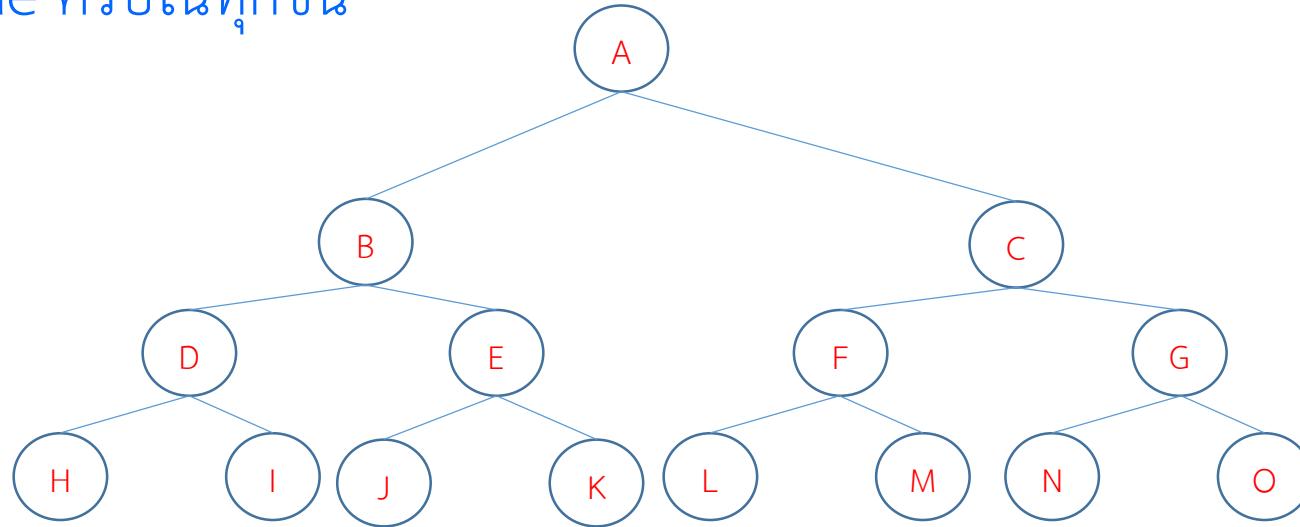
Trees:

ต้นไม้ที่วิภาคมีชื่อเรียกต่างกันตามคุณสมบัติของมัน

Trees:

Full Binary Tree

คือต้นไม้มี node ครบในทุกชั้น



จำนวน node ในแต่ละชั้นจะสามารถคำนวณได้

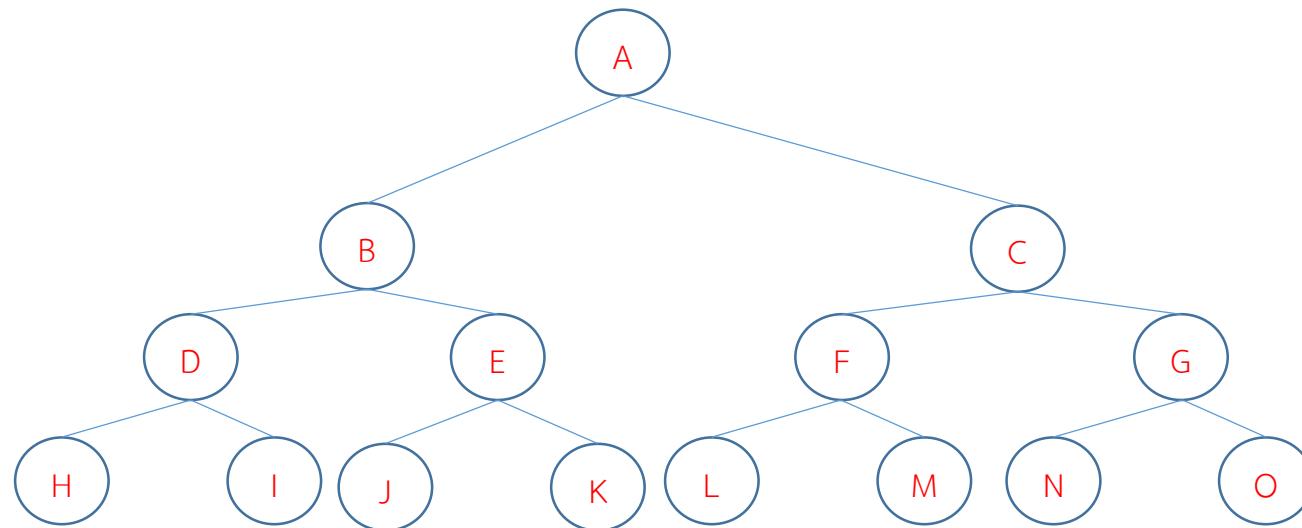
$$N = 2^{h+1} - 1$$

หรือหากรู้จำนวน node ทั้งหมด ก็สามารถคำนวณความสูงได้

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

Trees:

Full Binary Tree



ต้นไม้แบบ Full binary tree ที่มีจำนวน 15 node จะมีความสูงเท่าใด

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

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

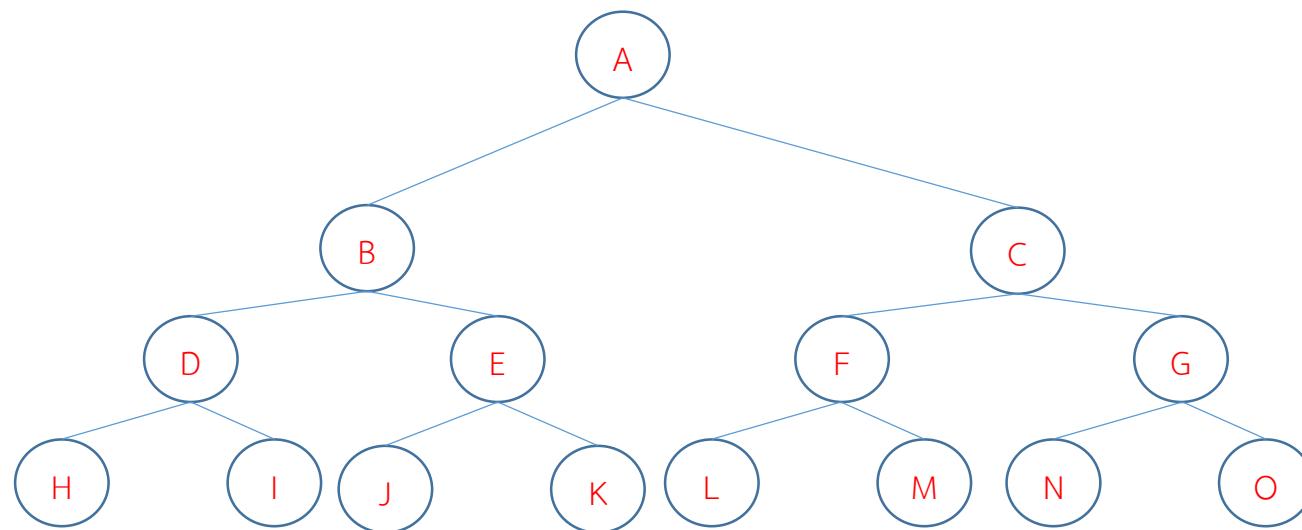
$$h = \log_2(16) - 1$$

$$h = 4 - 1$$

$$h = 3$$

Trees:

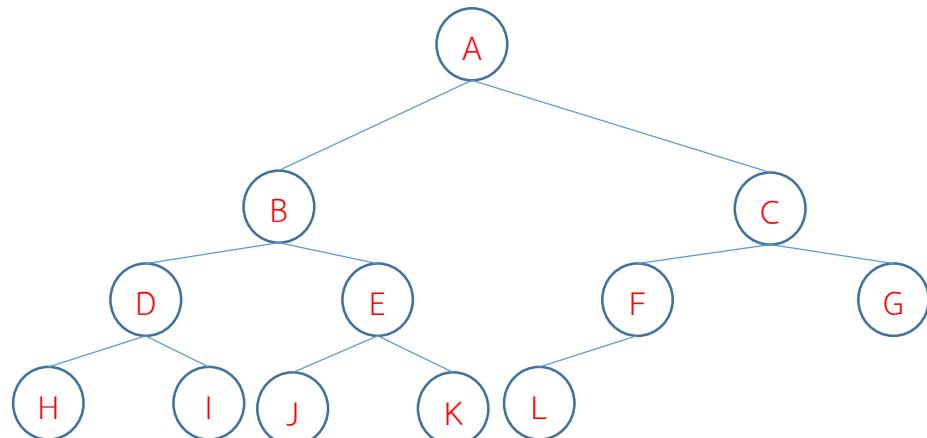
Full Binary Tree



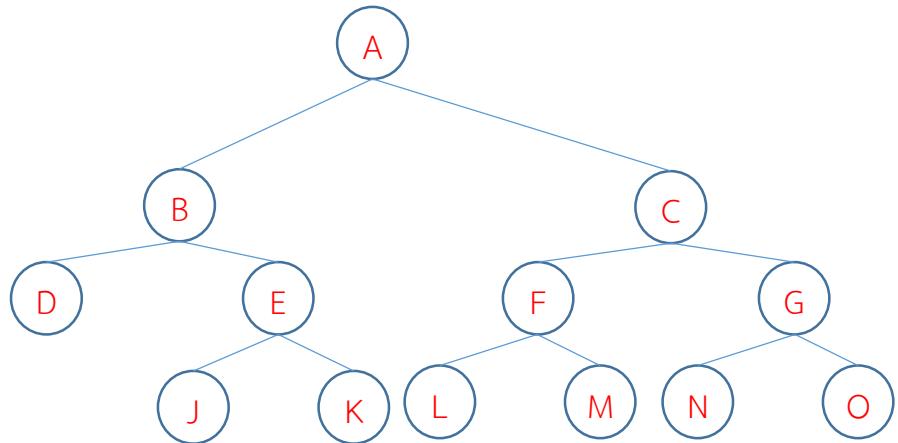
ต้นไม้แบบ Full binary tree มีโอกาสเกิดได้ยากในทางปฏิบัติ หากยอมให้ node ในชั้นล่างสุด ($h-1$) ไม่ครบได้ เฉพาะทางด้านขวา จะเรียกว่า Complete Binary Tree

Trees:

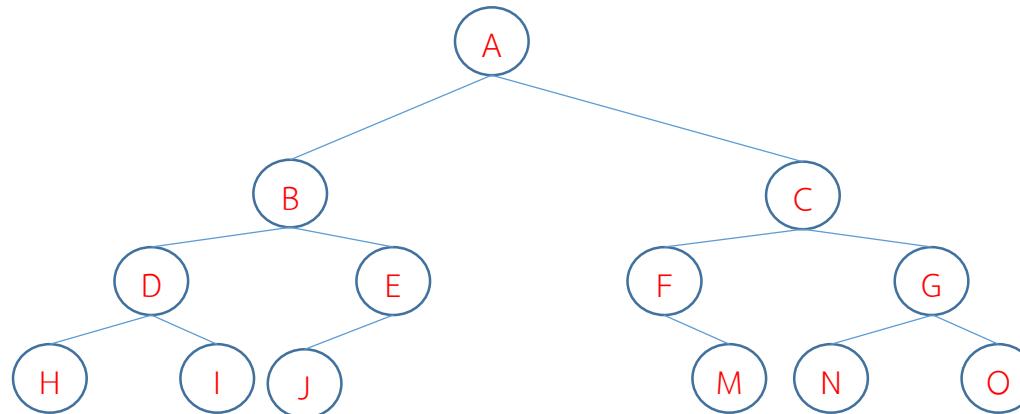
Complete Binary Tree



เป็น Complete Binary Tree



ไม่เป็น Complete Binary Tree



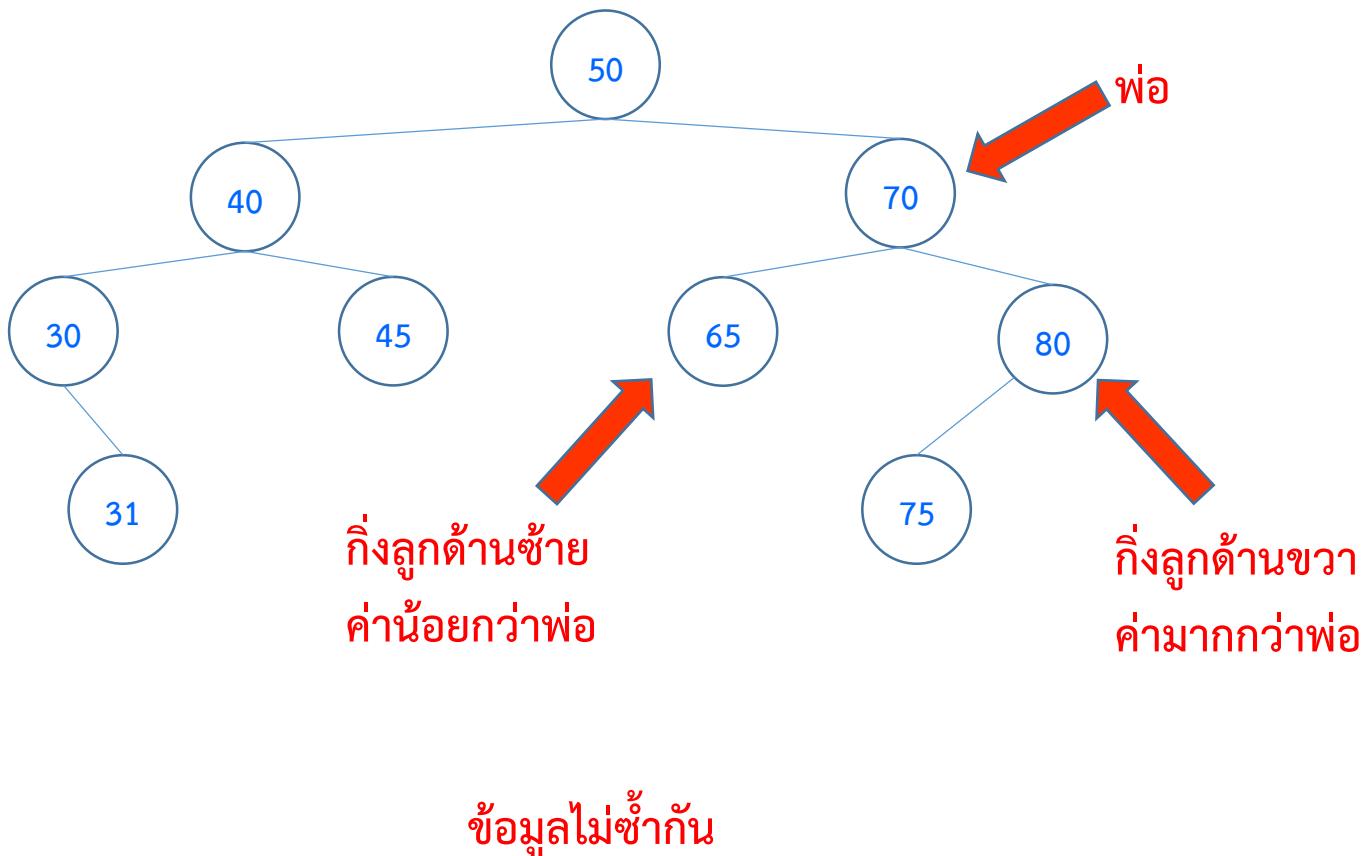
ไม่เป็น Complete Binary Tree

ต้นไม้ที่วิภาคมี 4 ประเภท

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

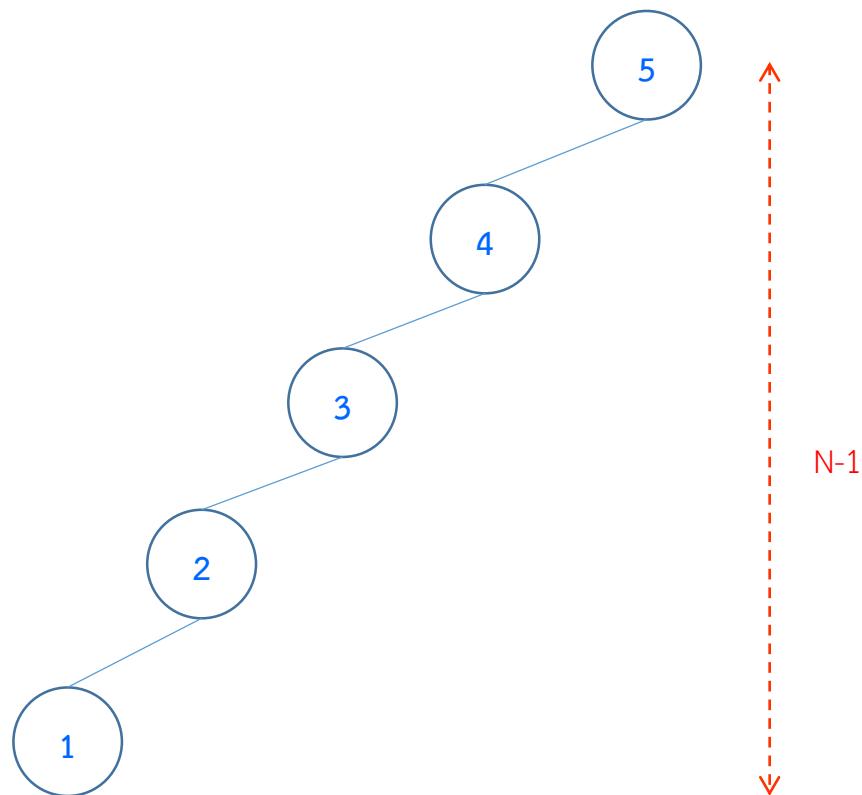
Binary Search Trees

- ค่าของ node จะซ้ำกันไม่ได้
- กิ่งลูกทางด้านซ้ายจะต้องมีค่าน้อยกว่าพ่อ และกิ่งลูกทางด้านขวาต้องมีค่ามากกว่าพ่อ เสมอ
- ข้อมูลในต้นไม้ต้องเป็นแบบที่สามารถเปรียบเทียบกันได้เท่านั้น



Binary Search Trees

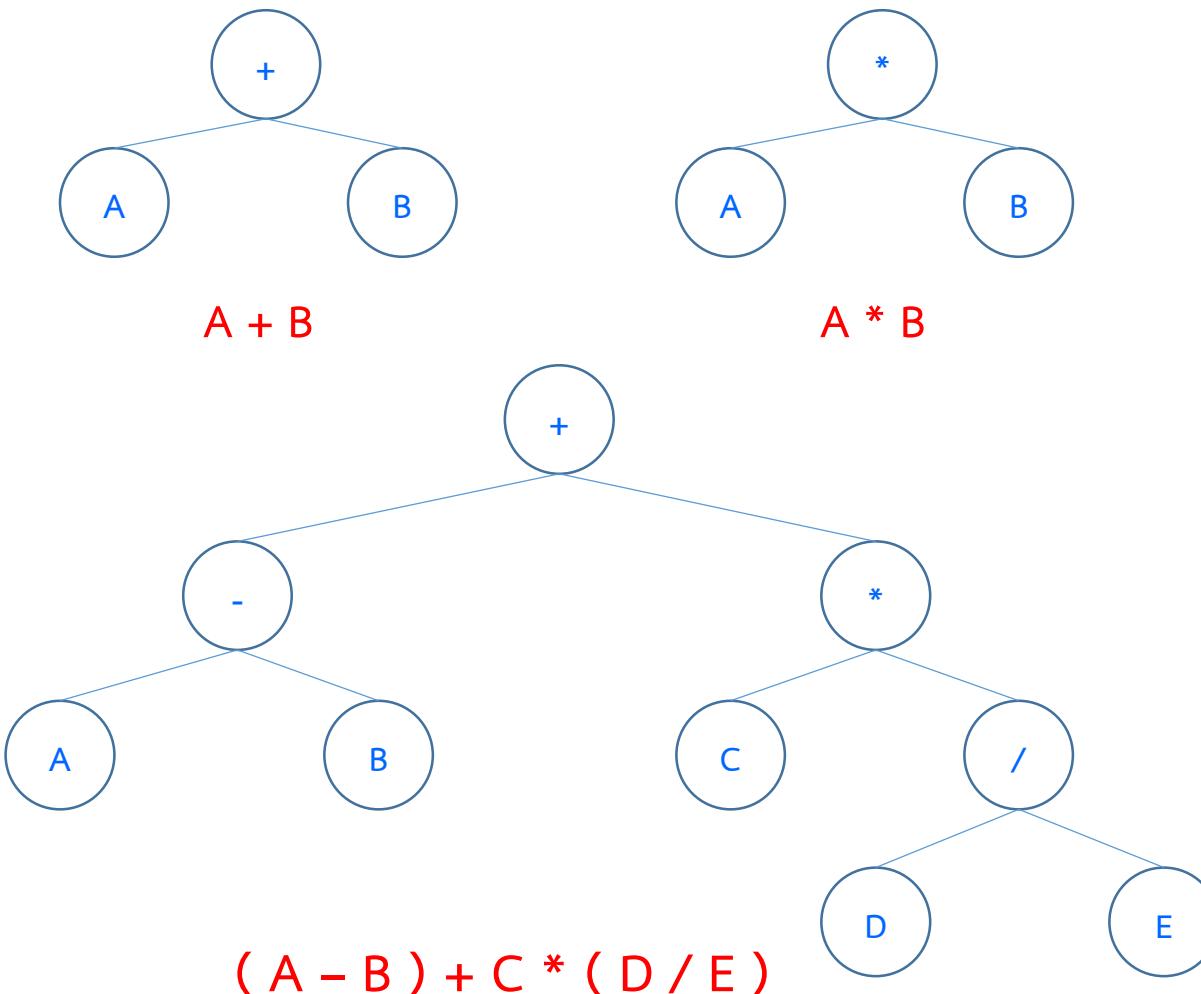
ต้นไม้แบบ Binary Search Trees จะสูงที่สุดไม่เกินเท่าใด ?



Trees: Expression Trees

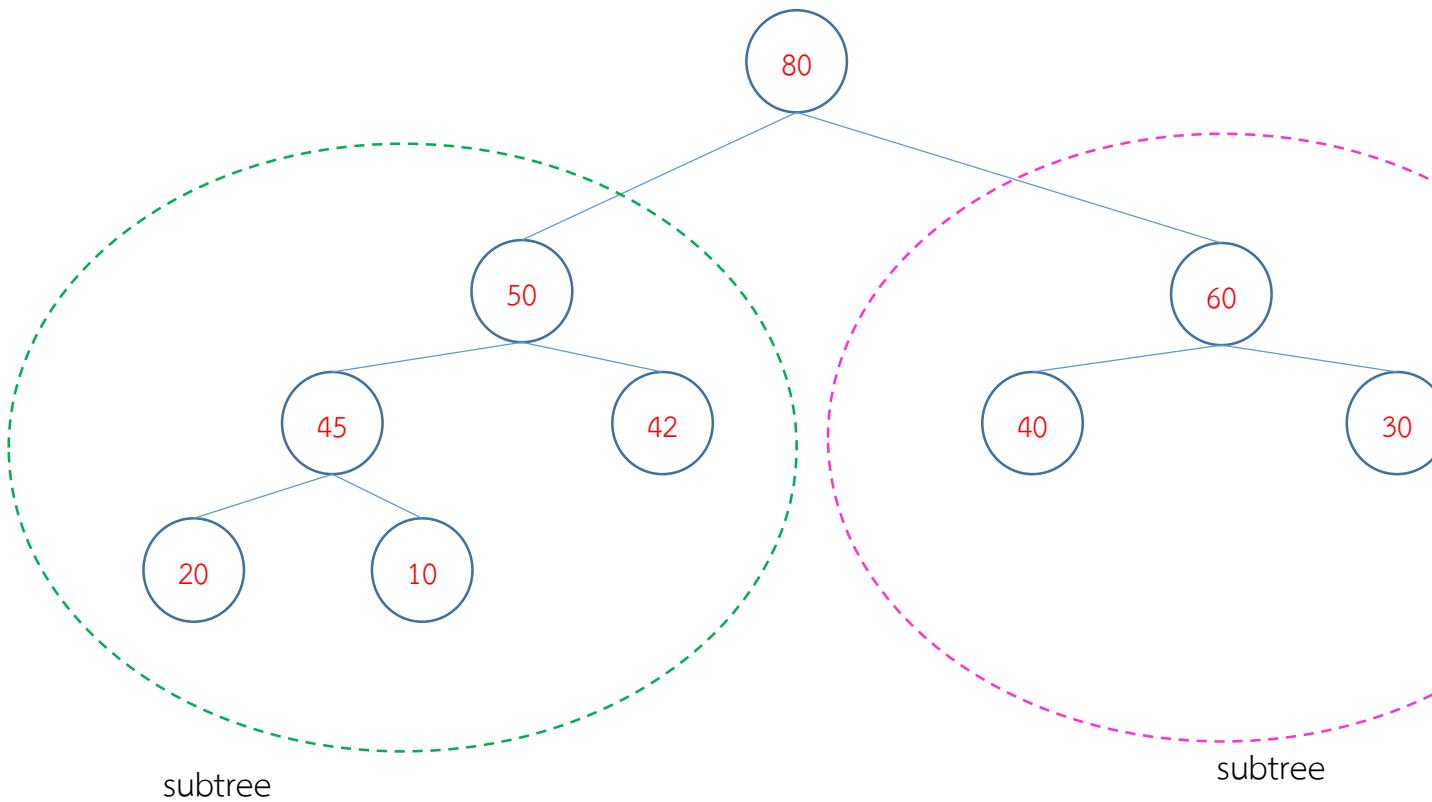
Expression Trees

- ใช้เก็บการประมวลผลทางคณิตศาสตร์ที่มีลำดับขั้น
- Operand จะอยู่ที่ปมใบเสมอ



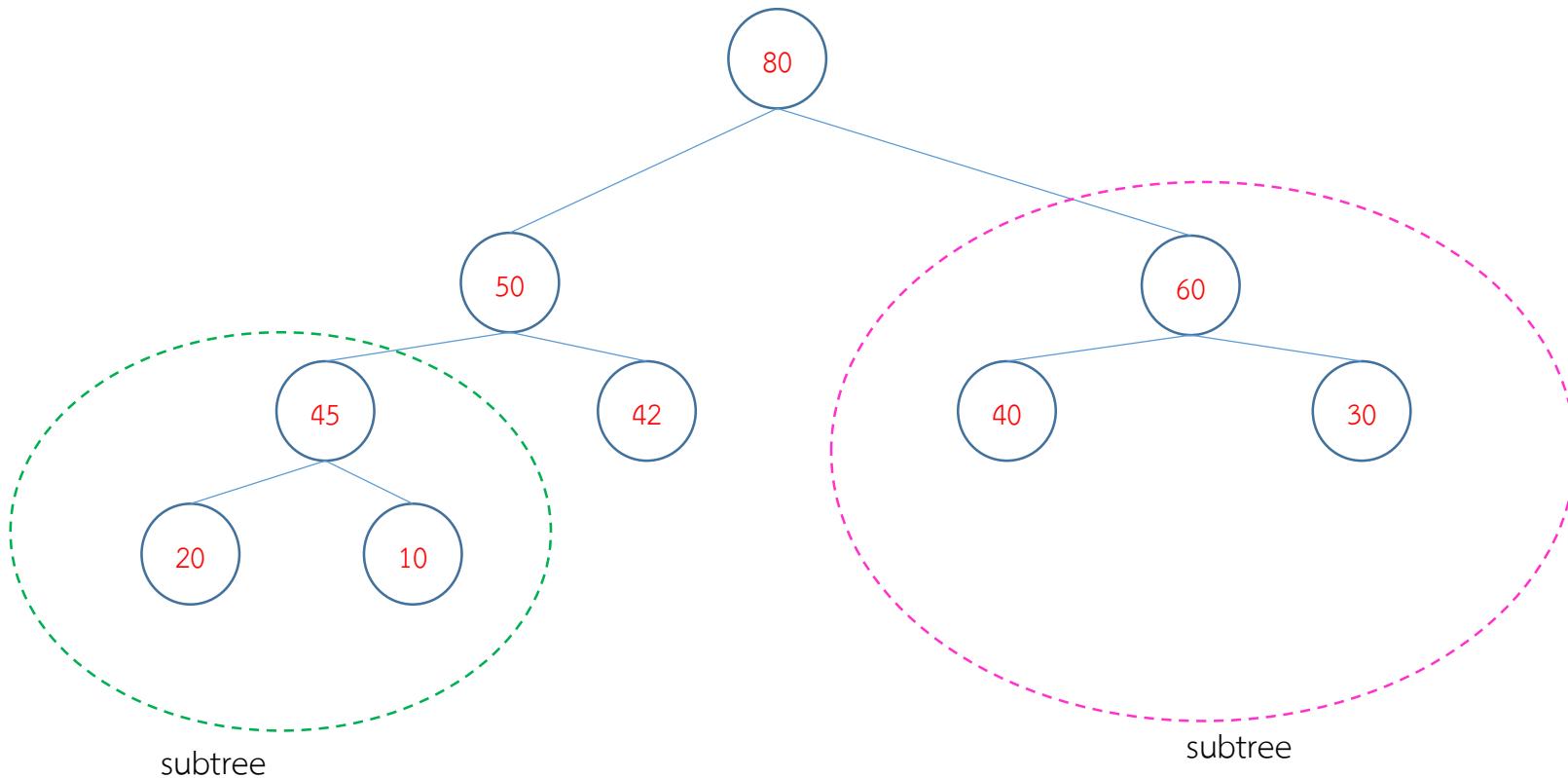
Heap Trees

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



Heap Trees

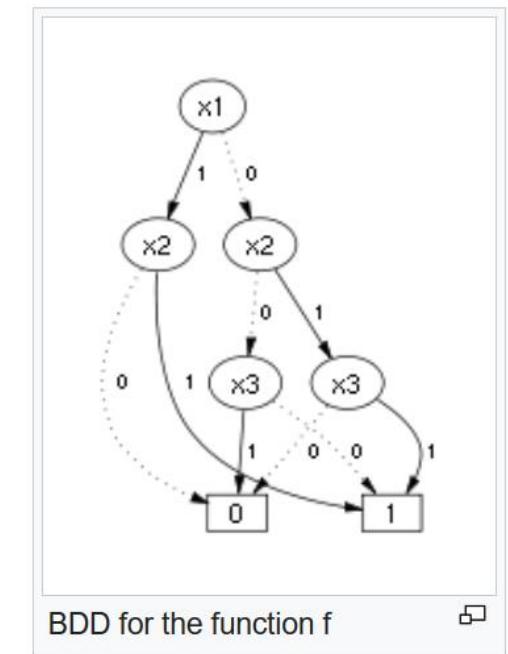
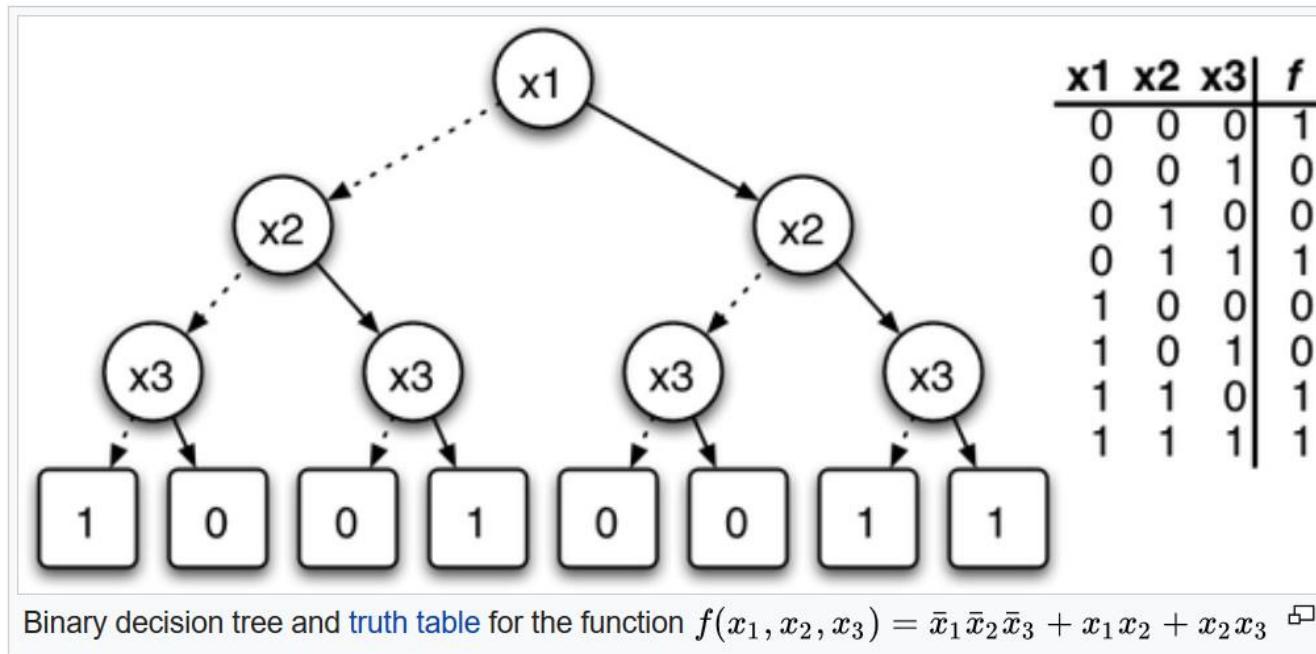
- คือ Binary tree แบบพิเศษ
- ค่าของ node ใด ๆ จะมากกว่าค่าใน subtree ทางด้านซ้ายและขวา เสมอ
- นิยมใช้สร้างอัลกอริทึมเรียงลำดับข้อมูล
- ใช้สร้าง Priority Queue



Trees: Binary Decision Trees

Binary Decision Trees

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



การสร้างต้นไม้

สิ่งที่ต้องคำนึง

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

วิธีที่ 1 ใช้ array ของ structure

สิ่งที่ต้องคำนึง

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

Trees: Array Structure Implementation

วิธีที่ 1 ใช้ array ของ structure

ข้อมูล	Link ไปยังลูกด้านซ้าย	Link ไปยังลูกด้านขวา
--------	-----------------------	----------------------

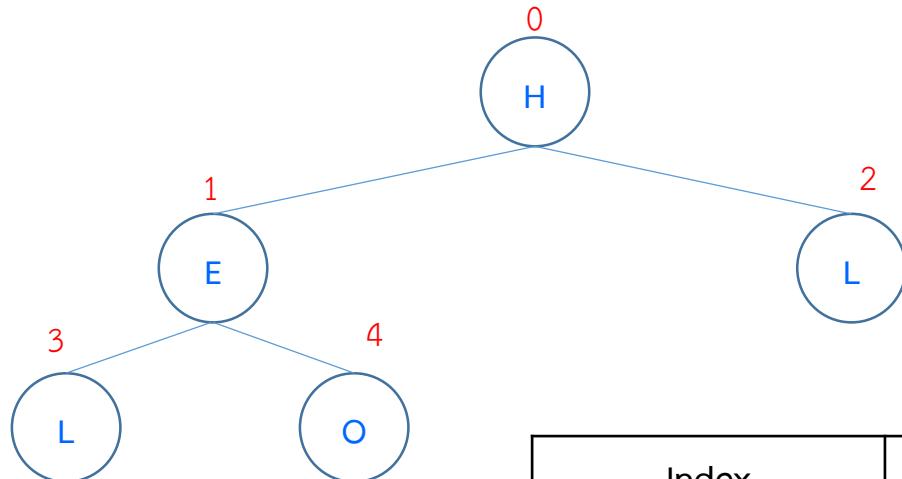
Struct

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

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

Trees: Implementation

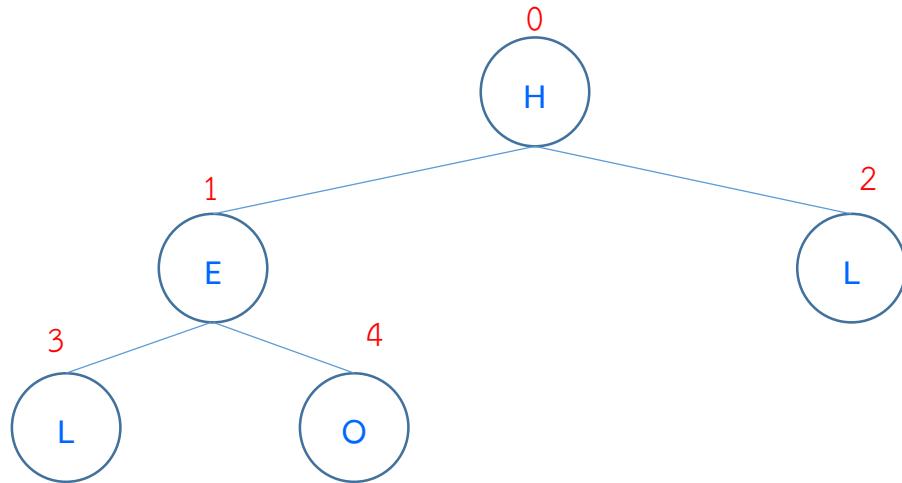
วิธีที่ 1 ใช้ array ของ structure



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

Trees: Implementation

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

```
#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;
}
```

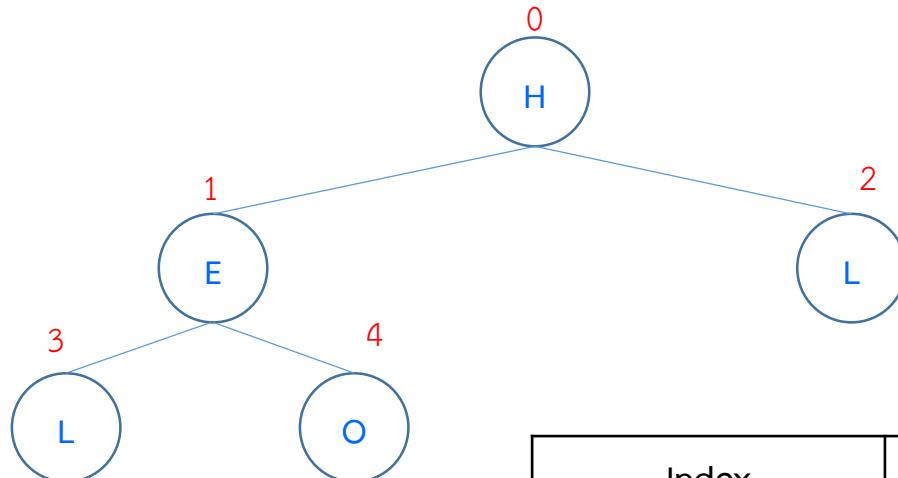
วิธีที่ 2 ใช้ array อาย่างเดียว

สิ่งที่ต้องคำนึง

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

Trees: Implementation

วิธีที่ 2 ใช้ array อาย่างเดียว



```
#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

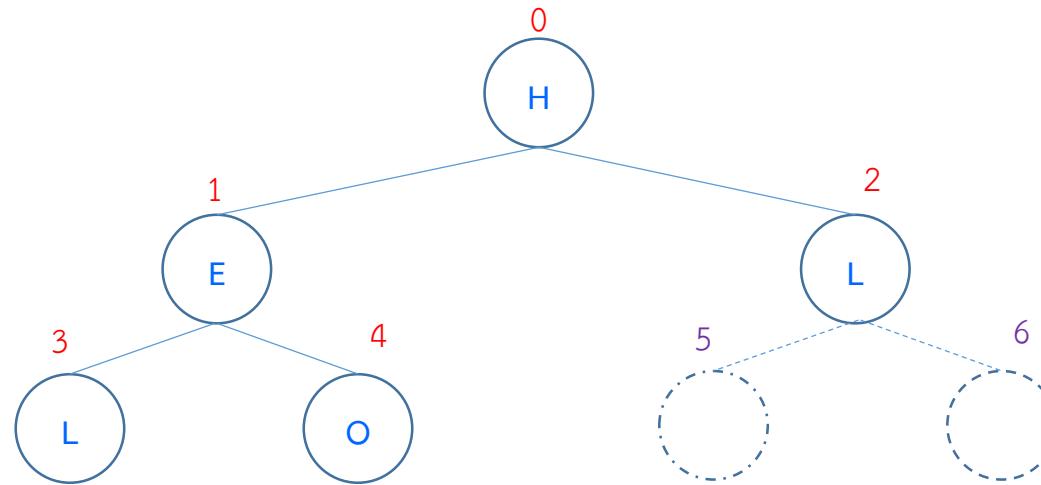
สิ่งที่ต้องคำนึง

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

Trees: Implementation

วิธีที่ 3 ใช้ array ตัวเดียว

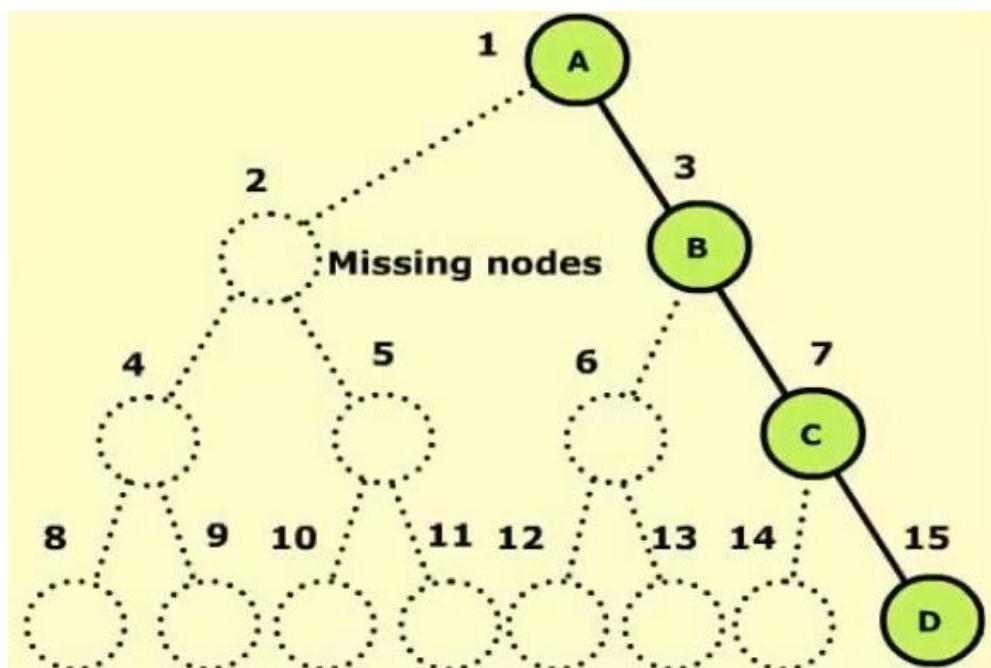
- วิธีนี้จะมองว่าต้นไม้เป็นแบบ Full binary tree
- หาก Node ในหน่วยไปก็จะให้ node นั้นเป็น Null



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

วิธีที่ 3 ใช้ array ตัวเดียว

- วิธีนี้จะมองว่าต้นไม้มีเป็นแบบ Full binary tree
- หาก Node ในหน้ายังไม่ถูกจัดให้ node นั้นเป็น Null
- เขียนโปรแกรมง่าย
- แต่หากต้นไม้มีลูปในทางทิศทางเดียว จะเปลี่ยนหน่วยความจำ



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
- หมายกับ complete tree
- หมายกับการค้นหาข้อมูล

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

Trees: Linked-list Implementation

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

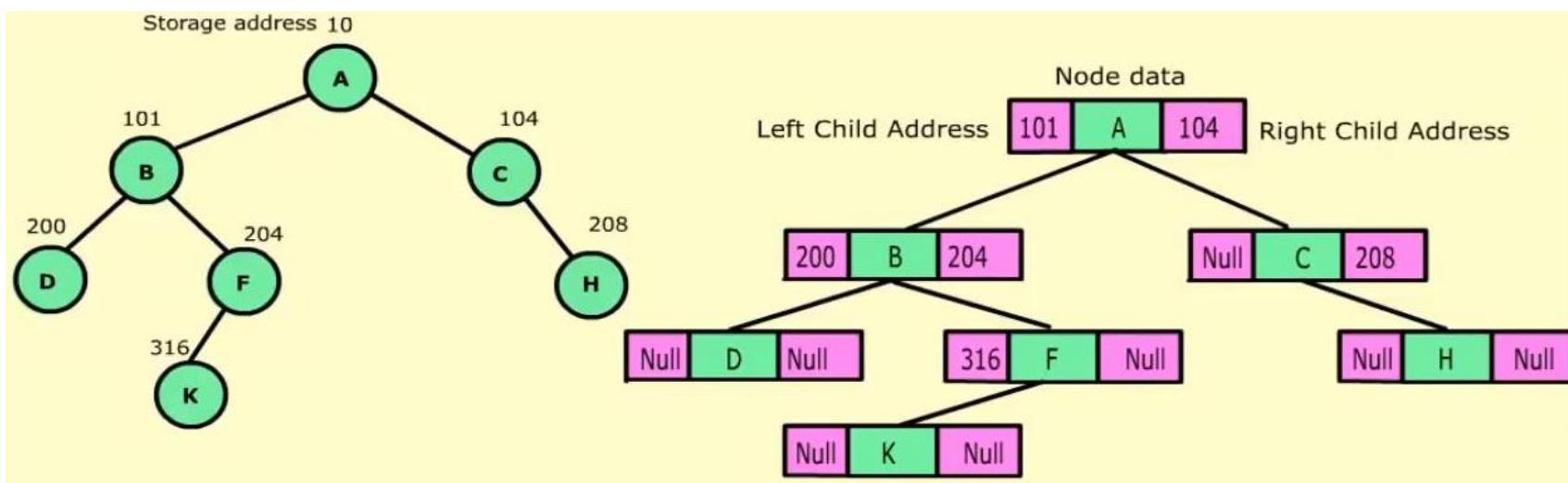
Trees: Linked-list Implementation

วิธีที่ 4 ใช้ Linked-list และ Structure

ข้อมูล	*Link ไปยังลูกด้านซ้าย	*Link ไปยังลูกด้านขวา
--------	------------------------	-----------------------

Struct

node ไหนไม่มีลูกก็กำหนดให้ link เป็น NULL

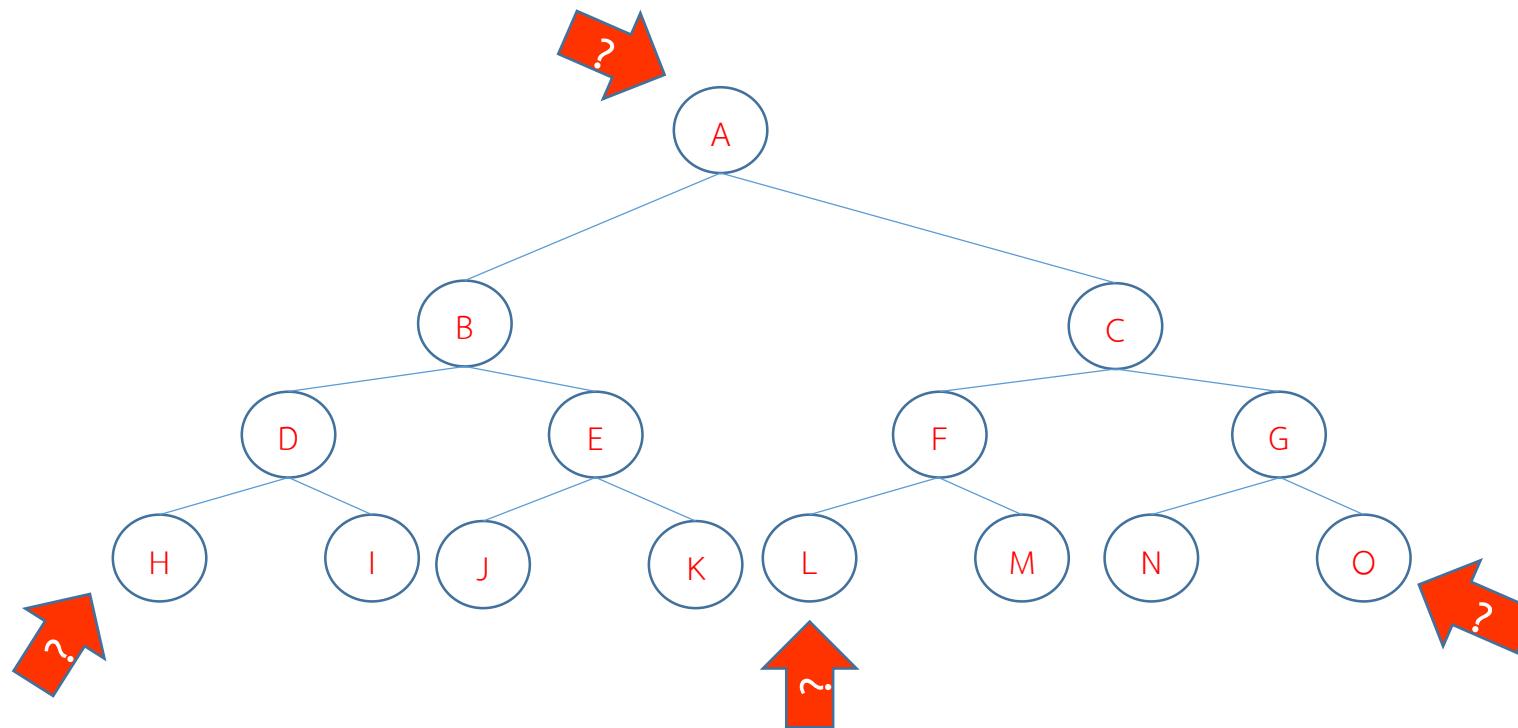


Linked-list ไม่สามารถเข้าถึงข้อมูลได้โดยตรงเหมือน Array

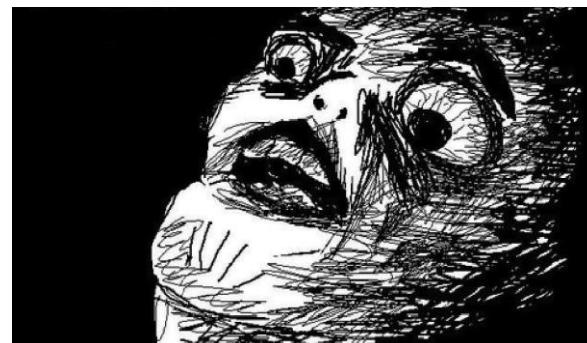
แก้ไขโครงสร้างจึงต้องใช้การ Traverse จาก Root

Trees: Traversal

การท่องเข้าไปในต้นไม้
จะเริ่มต้นจาก node ไหน ?



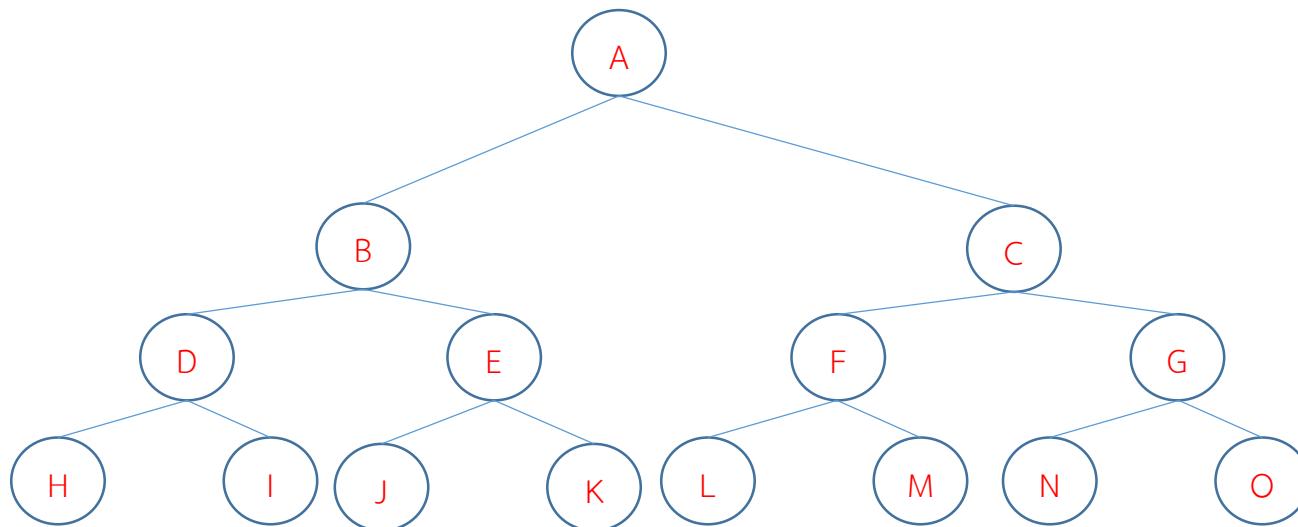
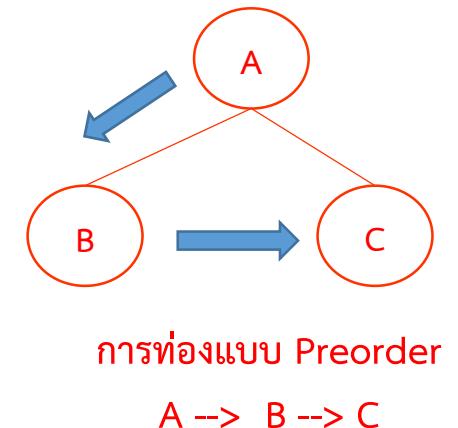
- 1) Preorder Traversal
- 2) Inorder Traversal
- 3) Postorder Traversal



Trees: Traversal

Preorder Traversal

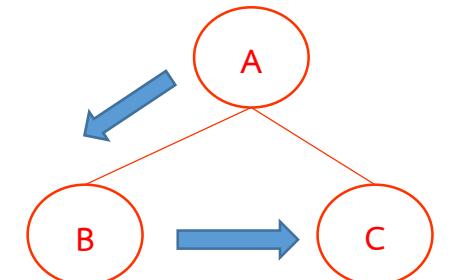
- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



Trees: Traversal

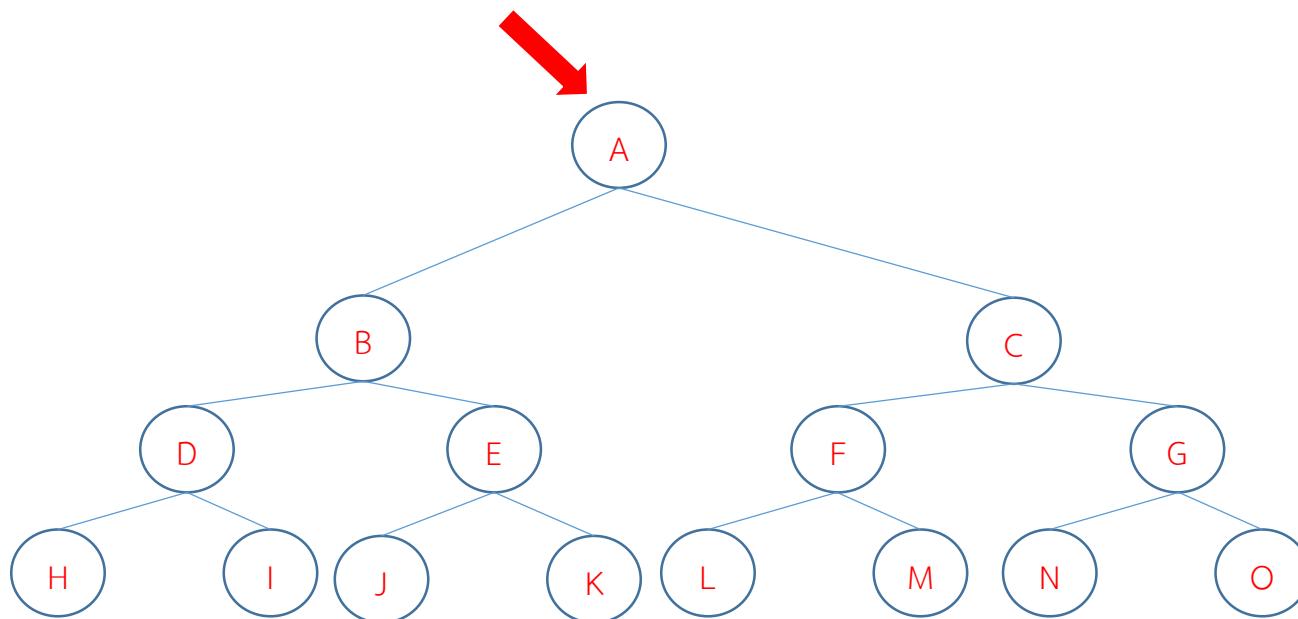
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

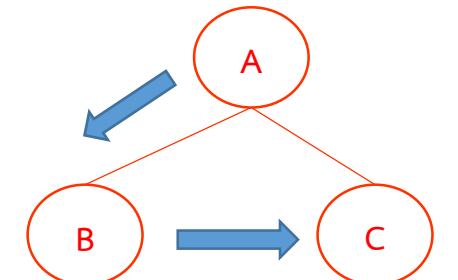
A --> B --> C



Trees: Traversal

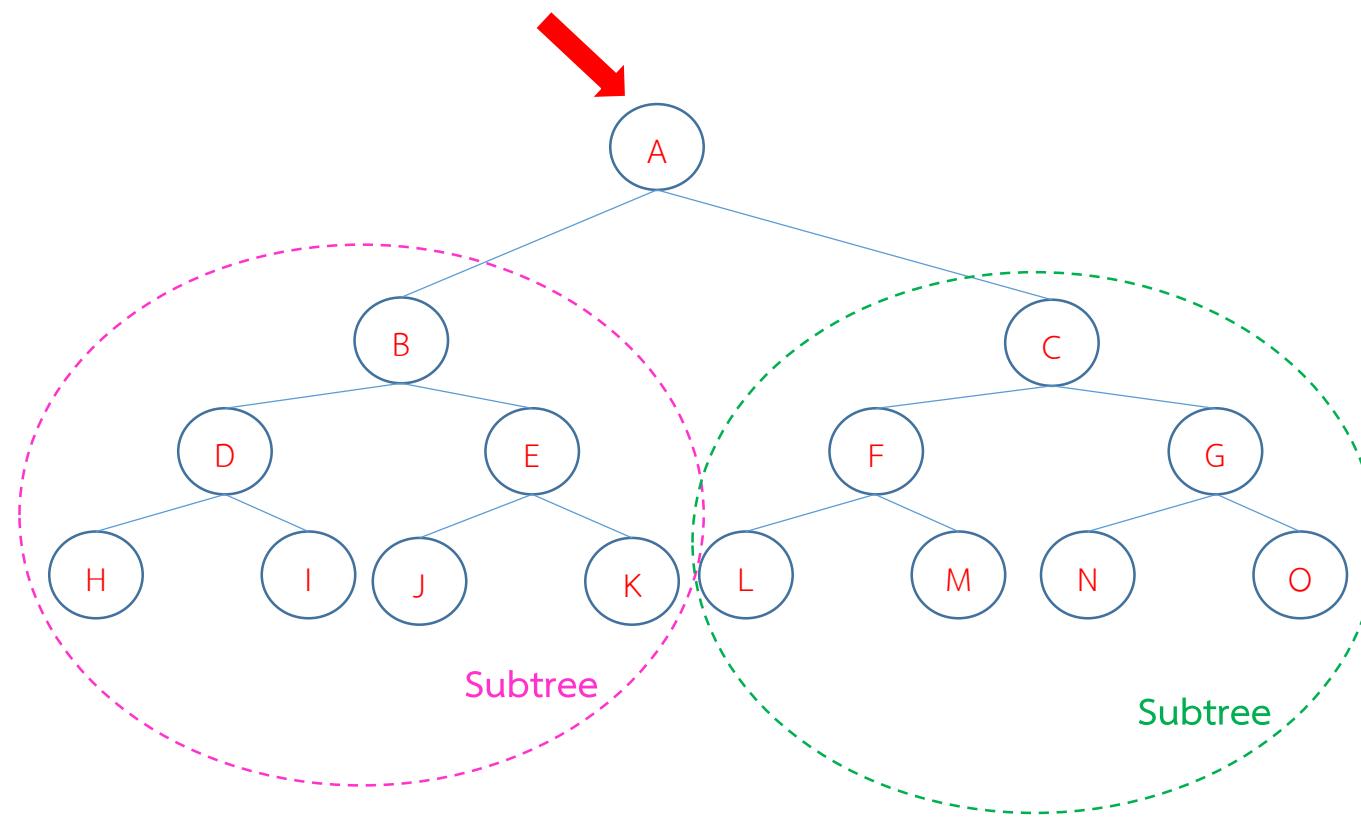
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

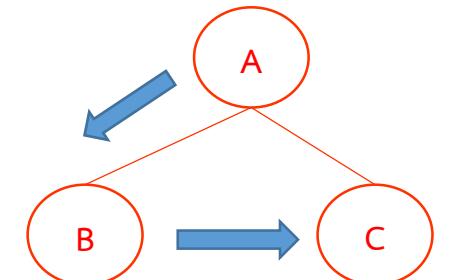
A --> B --> C



Trees: Traversal

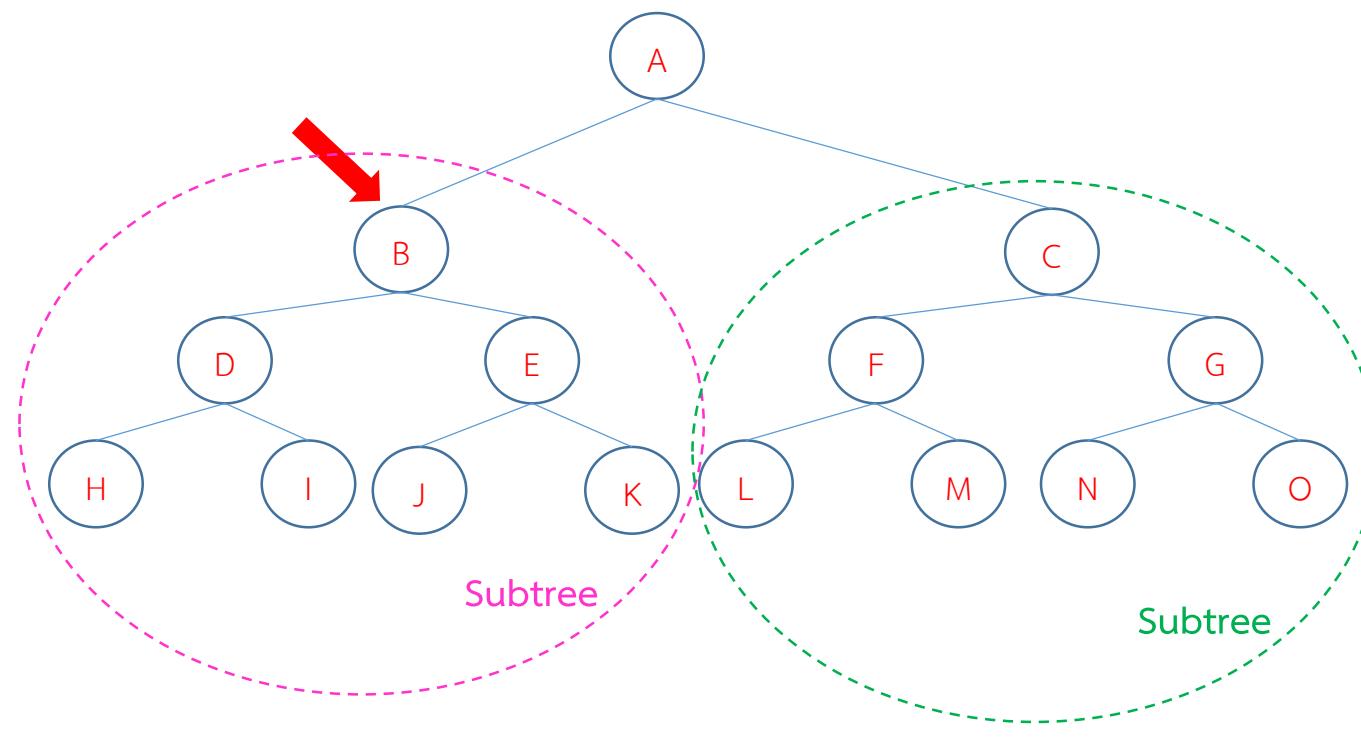
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

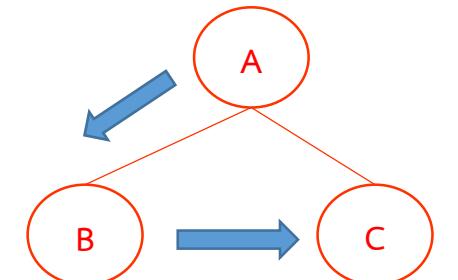
A --> B --> C



Trees: Traversal

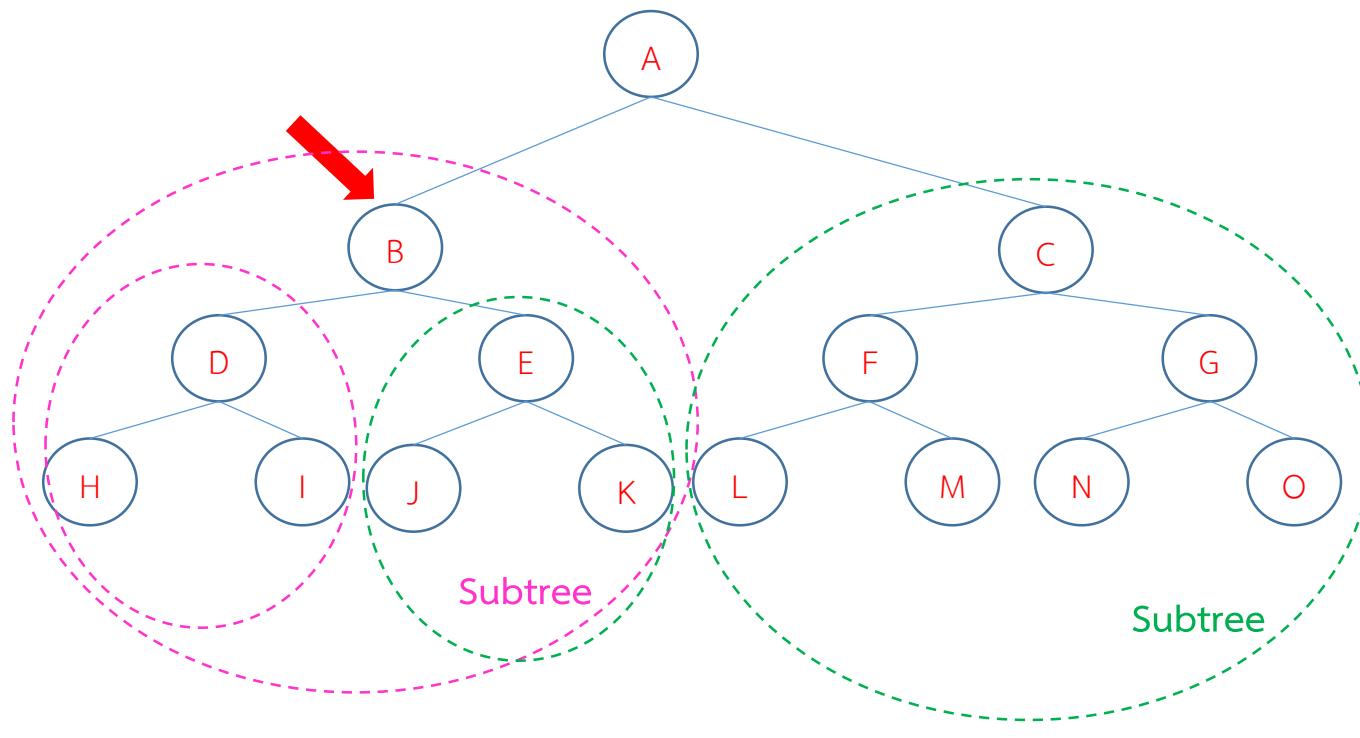
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

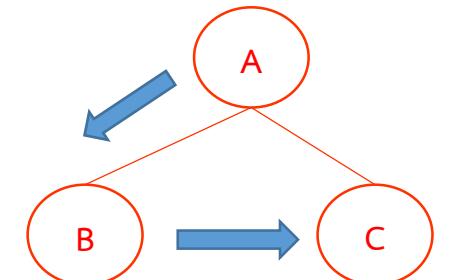
A --> B --> C



Trees: Traversal

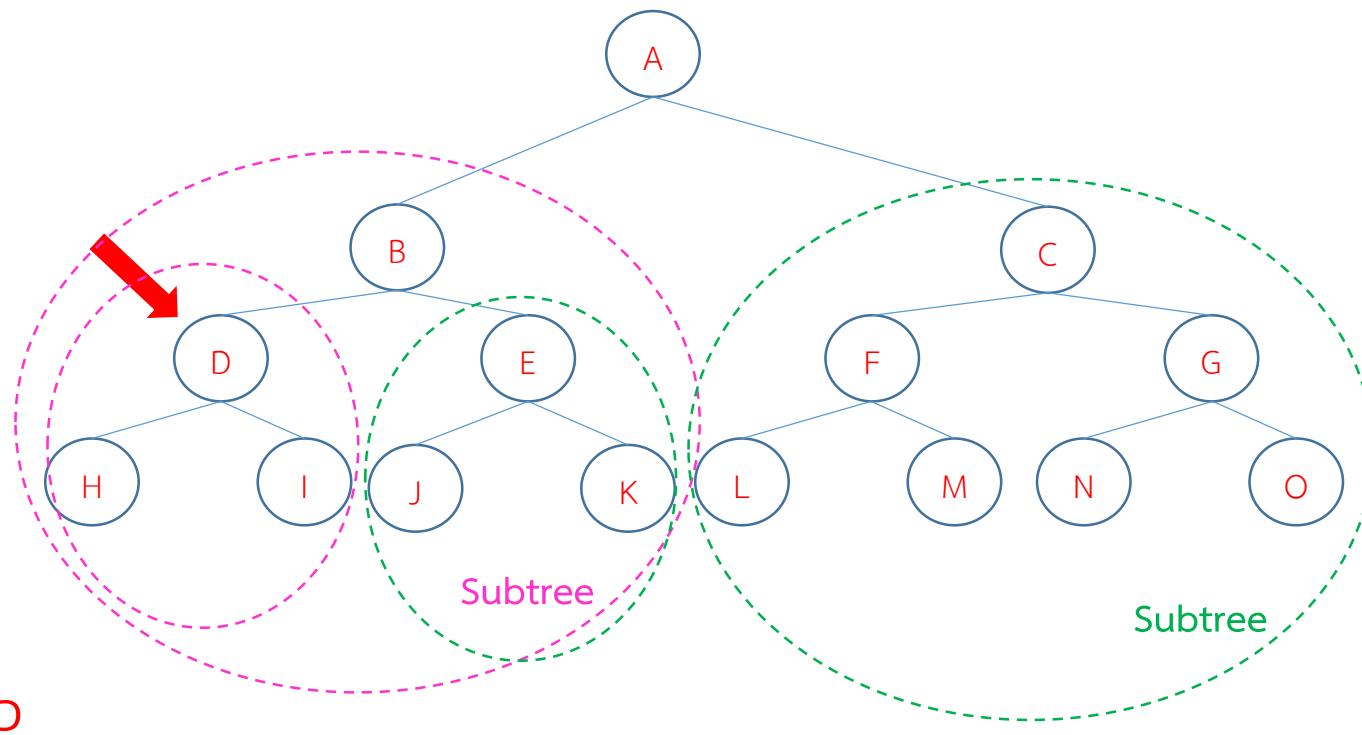
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

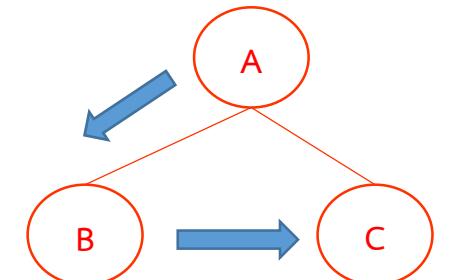
A --> B --> C



Trees: Traversal

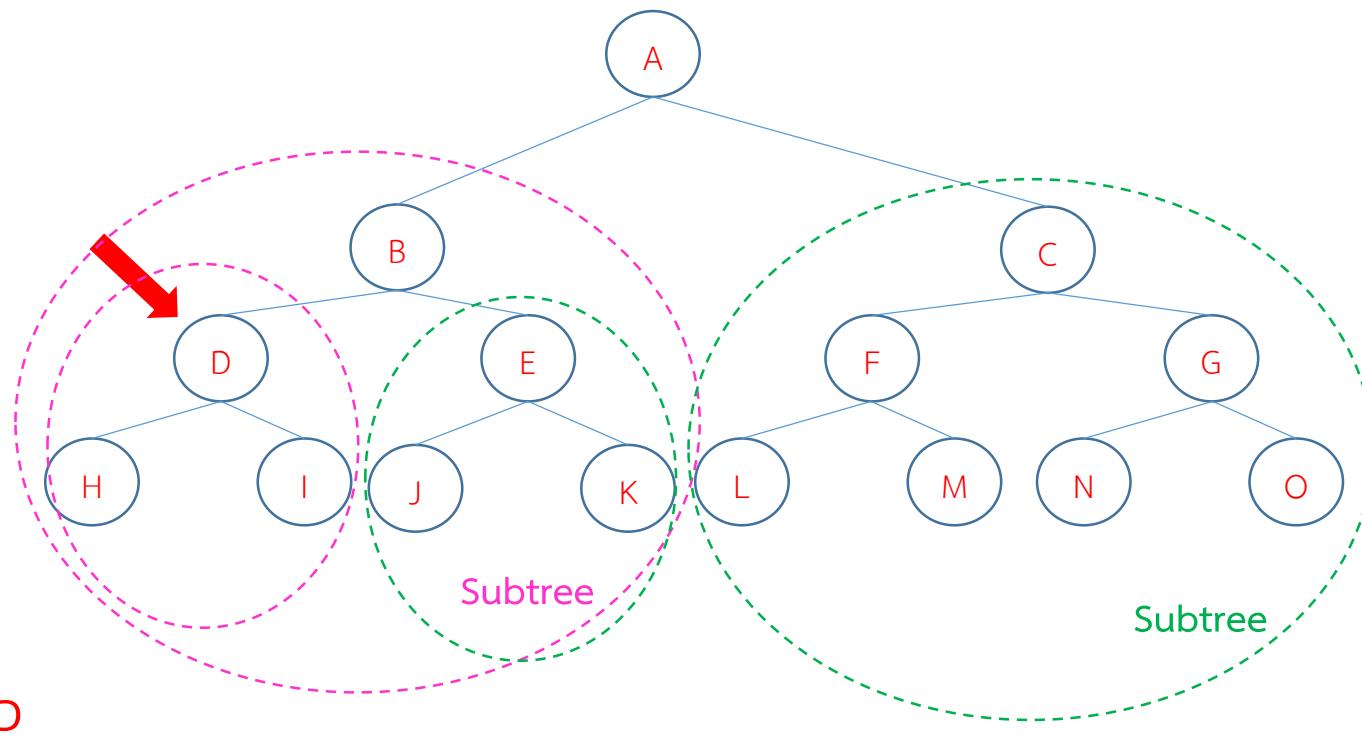
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

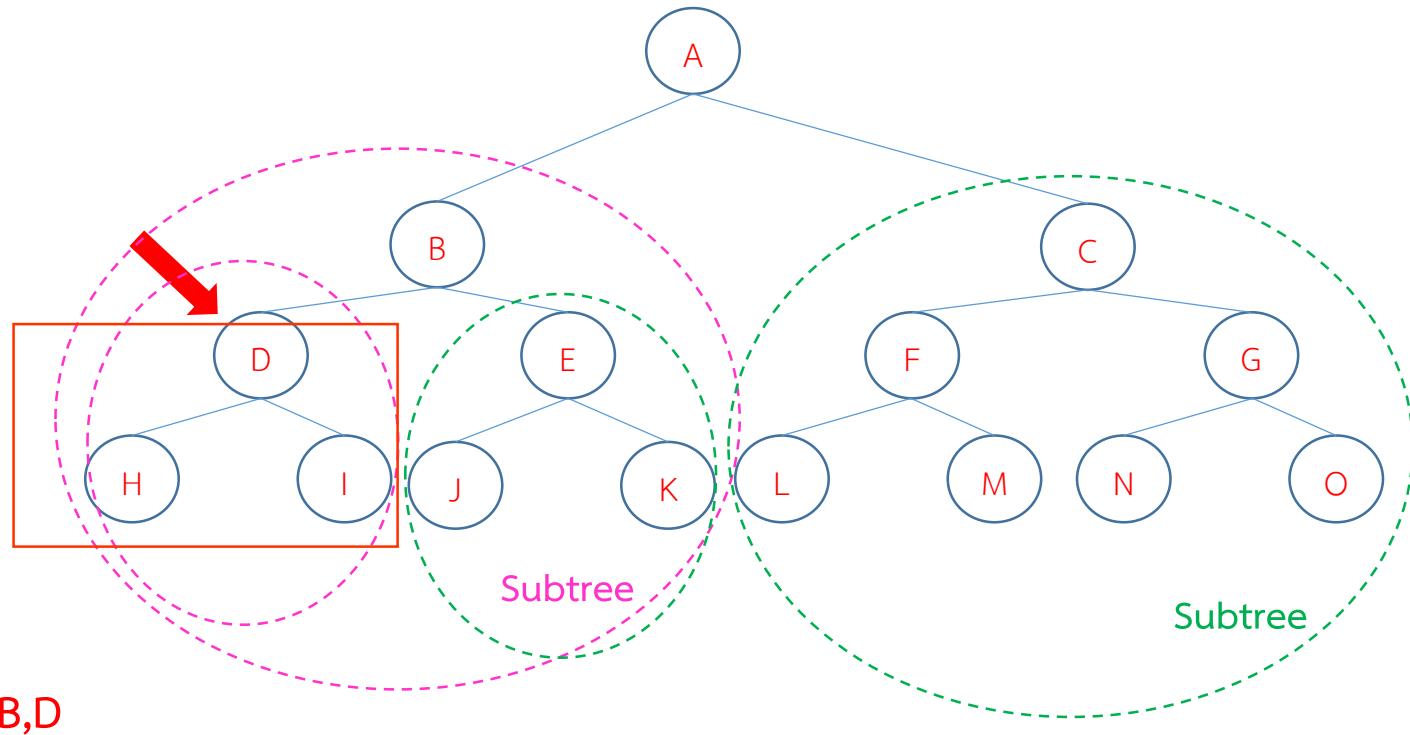
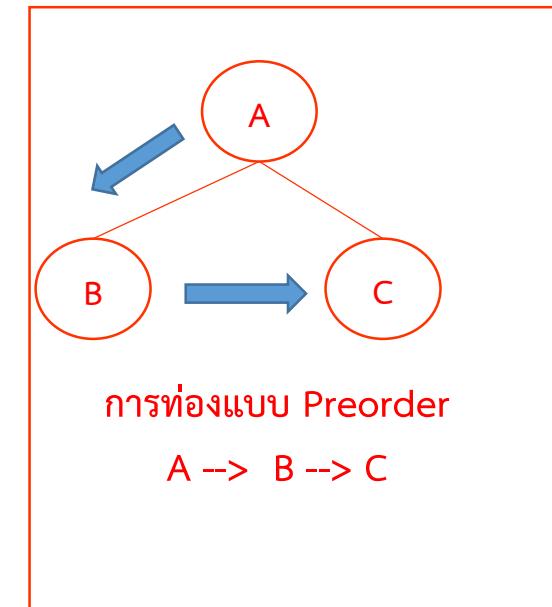
A --> B --> C



Trees: Traversal

Preorder Traversal

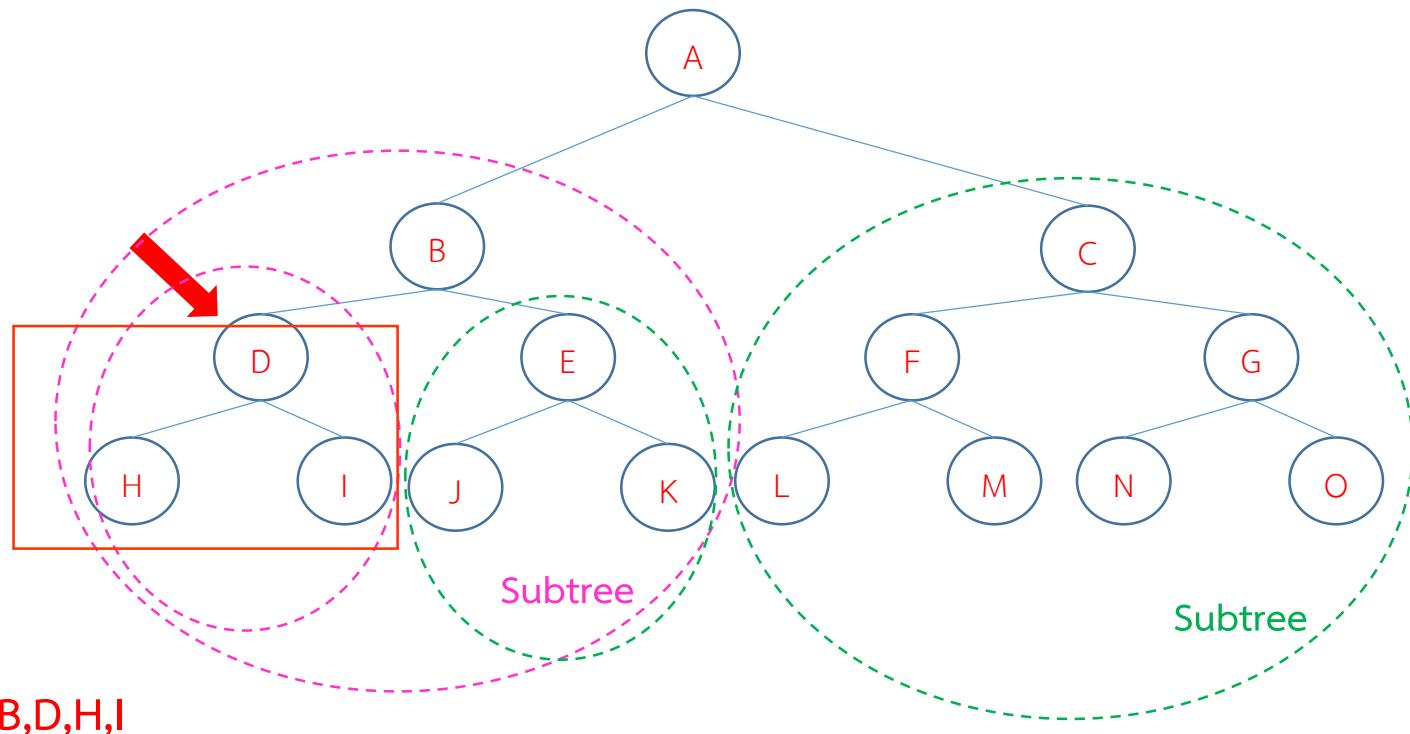
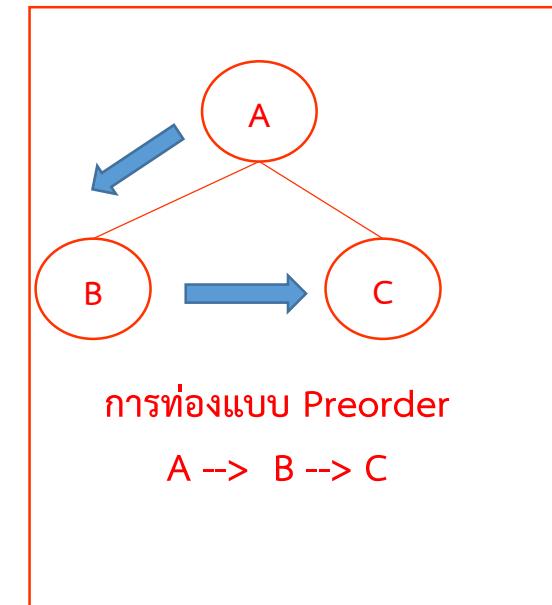
- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



Trees: Traversal

Preorder Traversal

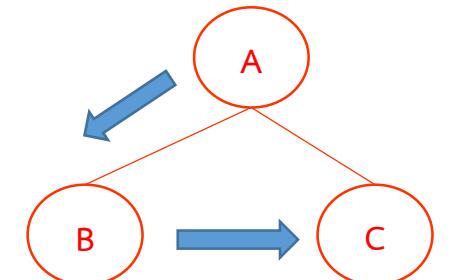
- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



Trees: Traversal

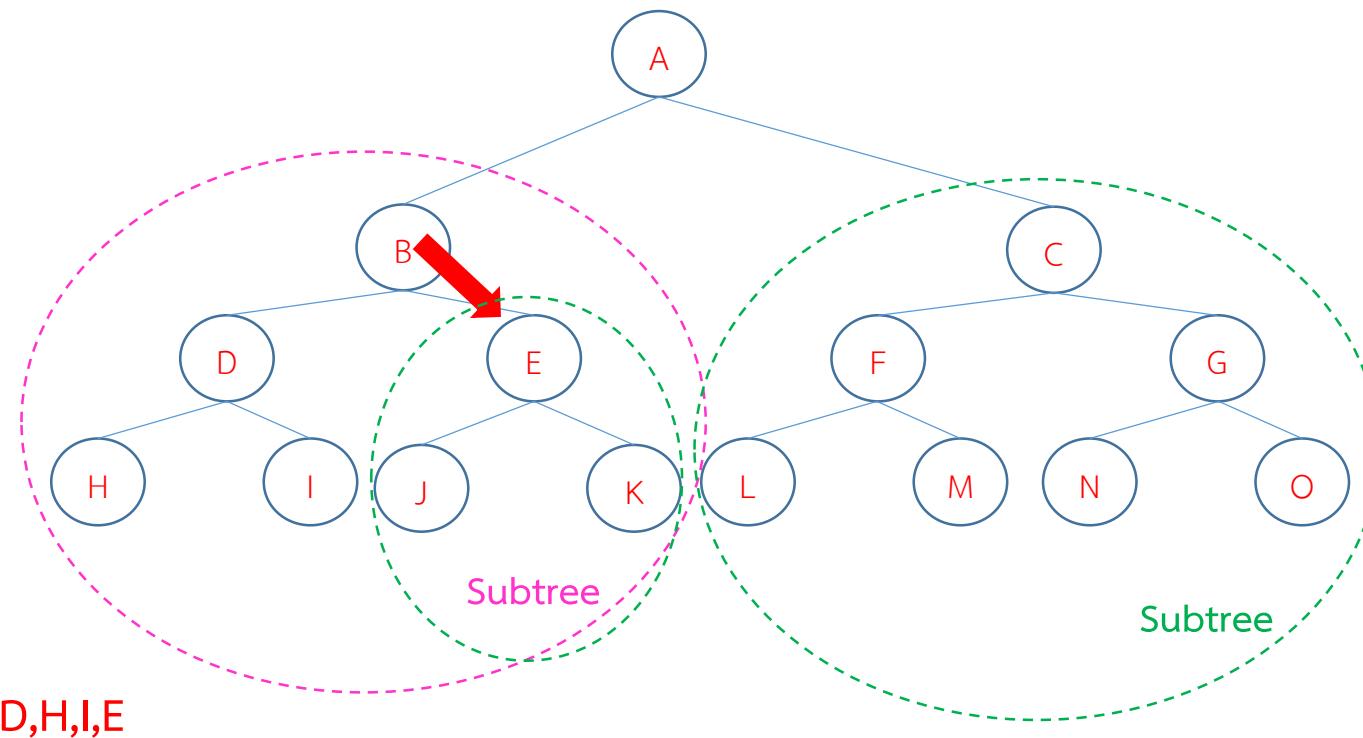
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถ่ไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถ่ไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

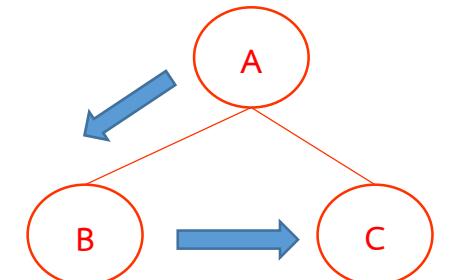
A --> B --> C



Trees: Traversal

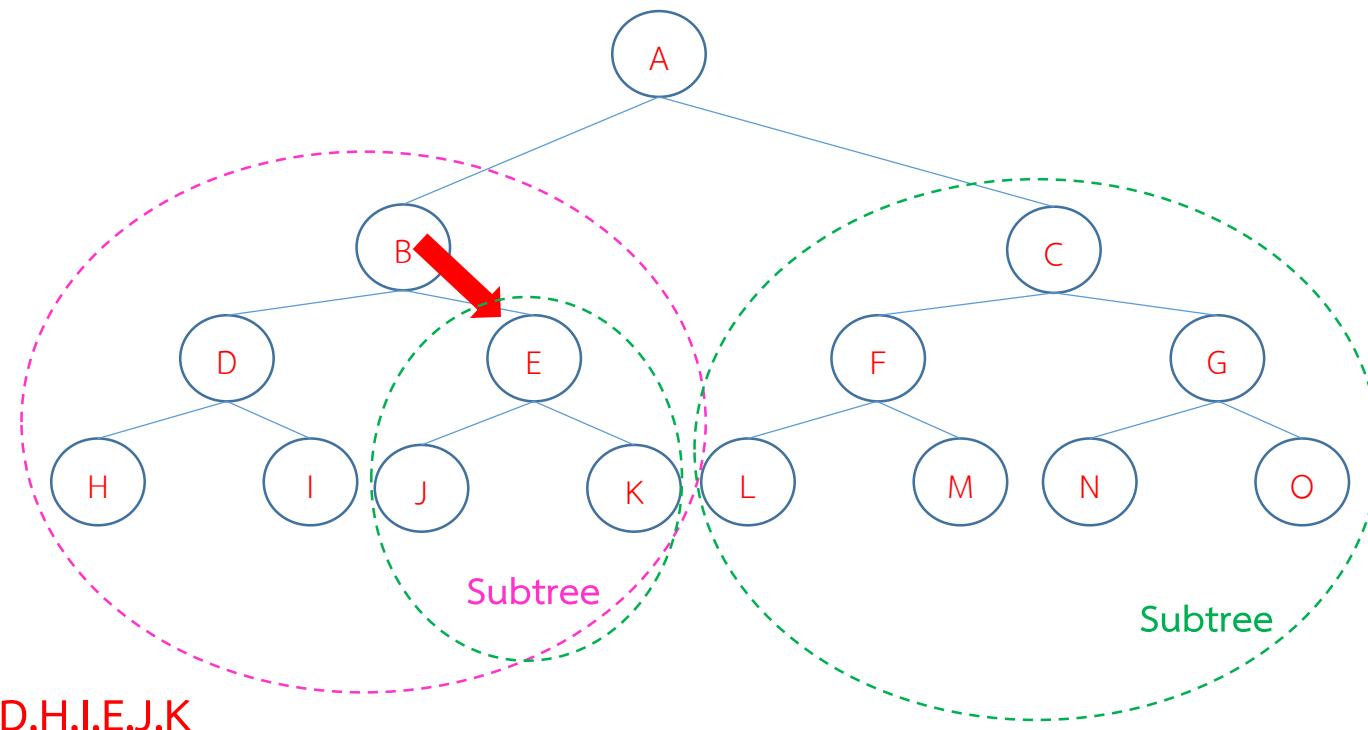
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

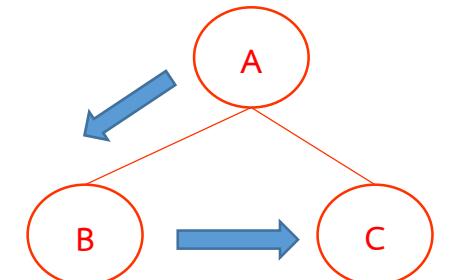
A --> B --> C



Trees: Traversal

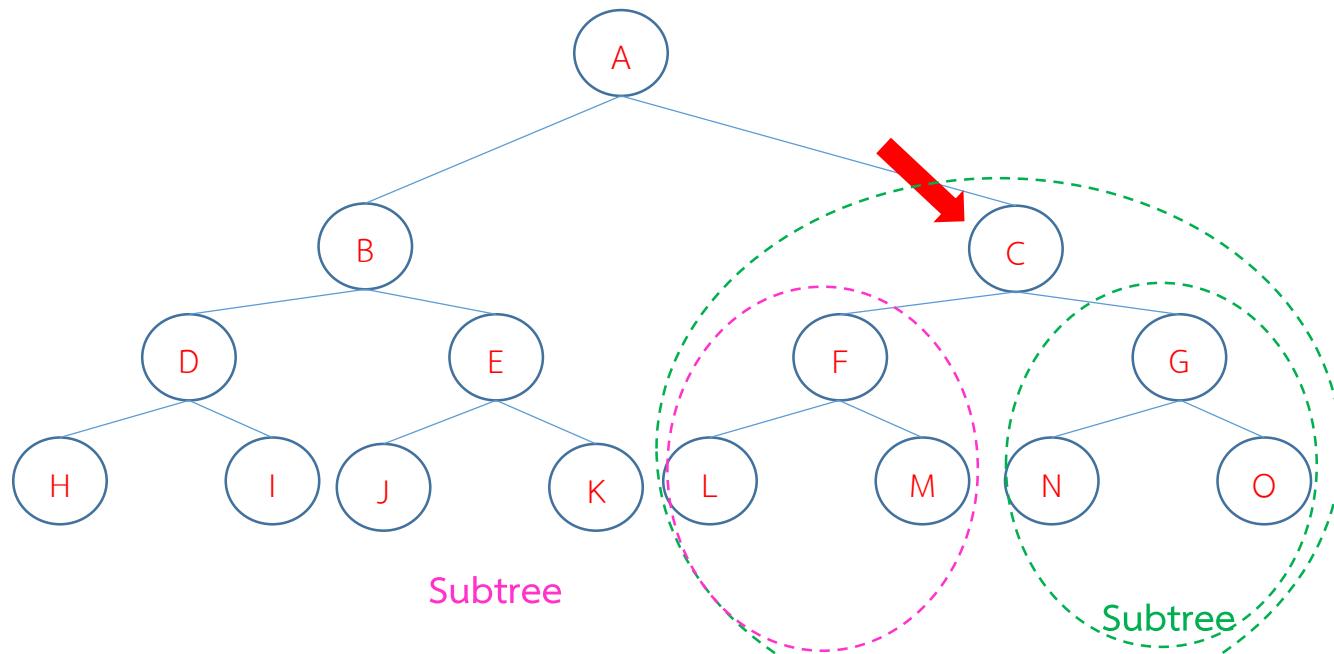
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

A --> B --> C

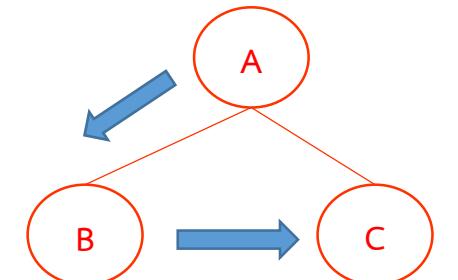


A,B,D,H,I,E,J,K,C

Trees: Traversal

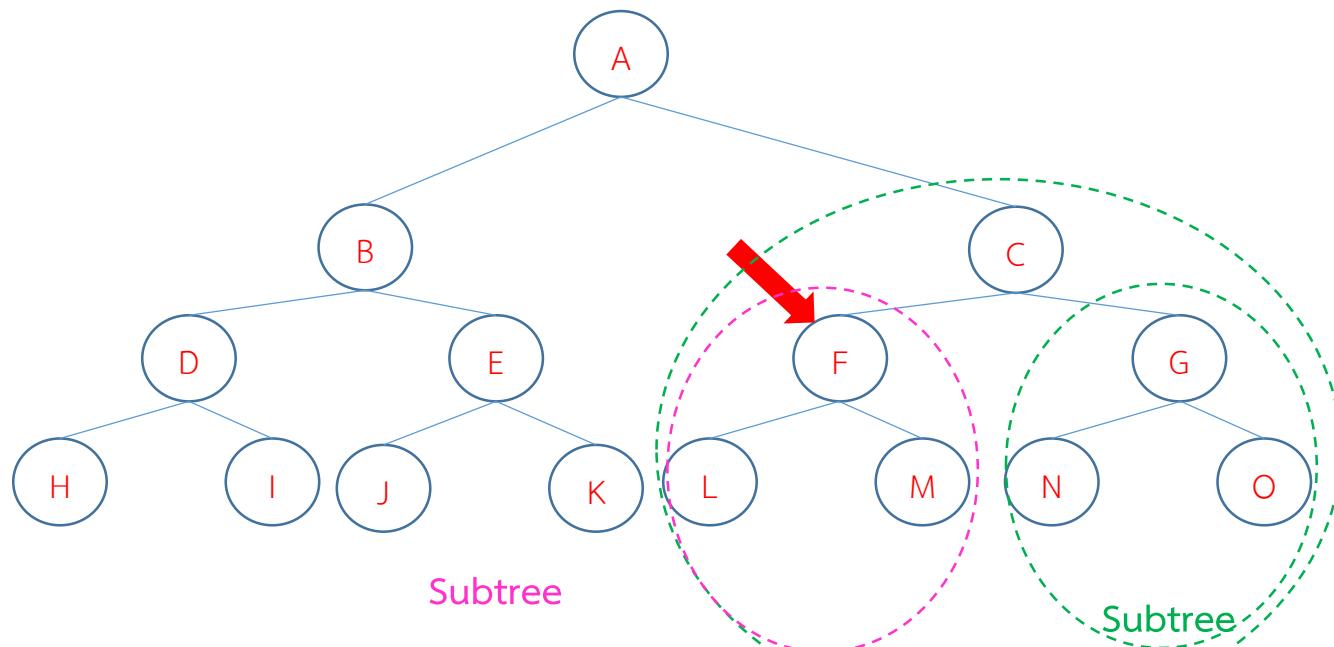
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

A --> B --> C

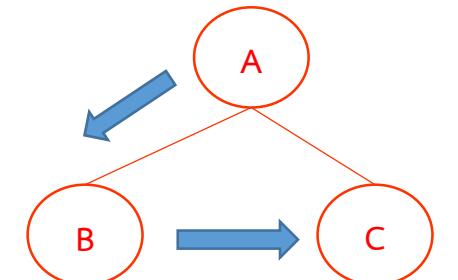


A,B,D,H,I,E,J,K,C,F,L,M

Trees: Traversal

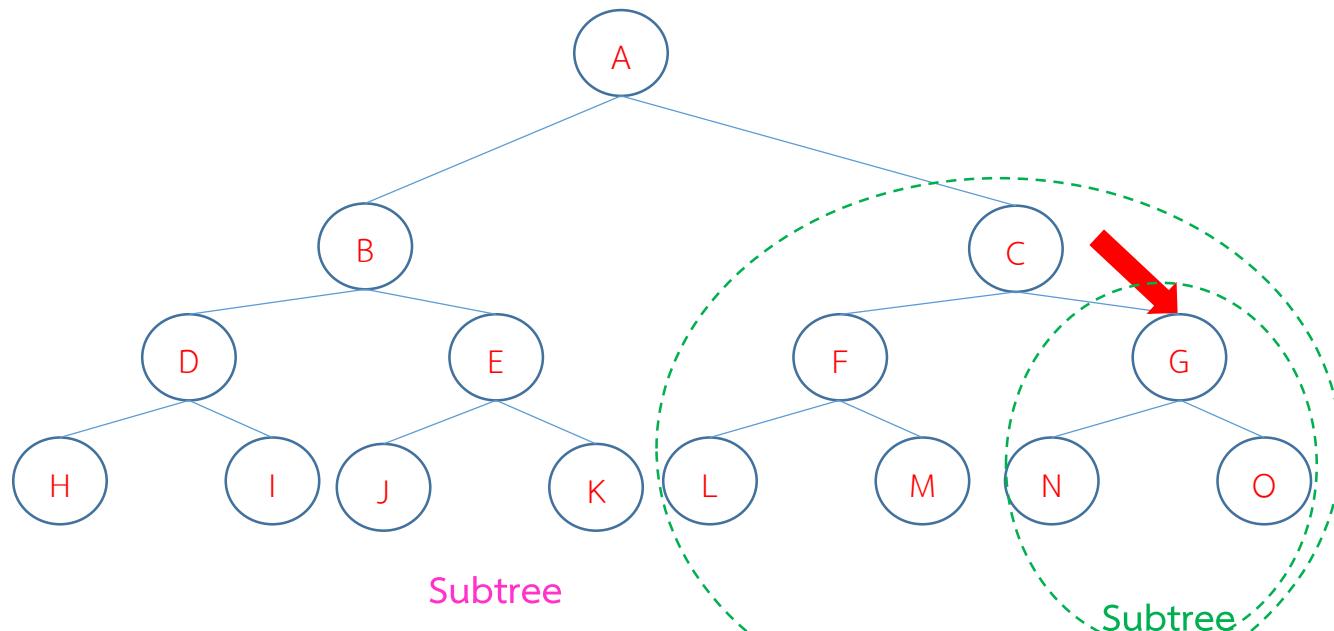
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

A --> B --> C

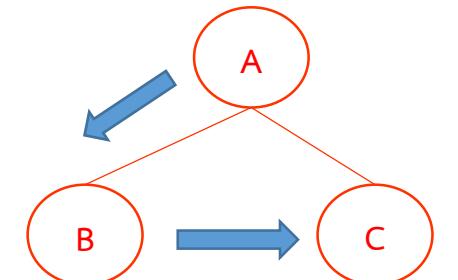


A,B,D,H,I,E,J,K,C,F,L,M,G,N,O

Trees: Traversal

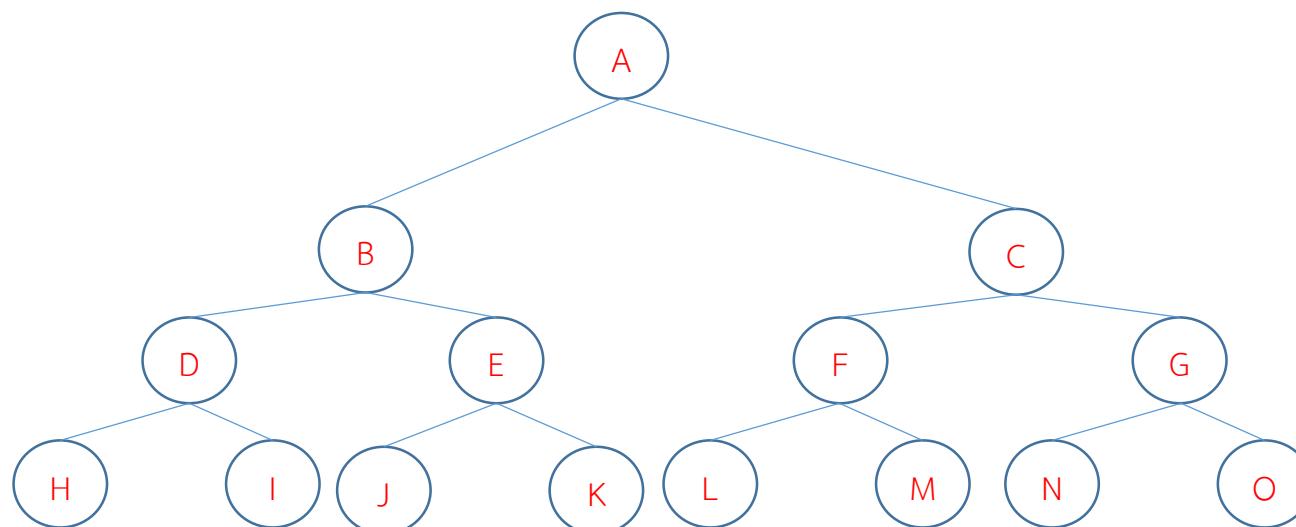
Preorder Traversal

- 1) เริ่มจาก Root Node
- 2) ไถไปทาง subtree ด้านซ้ายแบบ Preorder
- 3) ไถไปทาง subtree ด้านขวาแบบ Preorder



การท่องแบบ Preorder

A --> B --> C

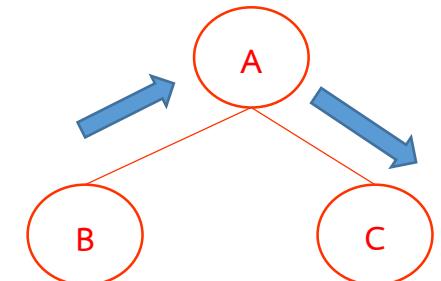


A,B,D,H,I,E,J,K,C,F,L,M,G,N,O

Trees: Traversal

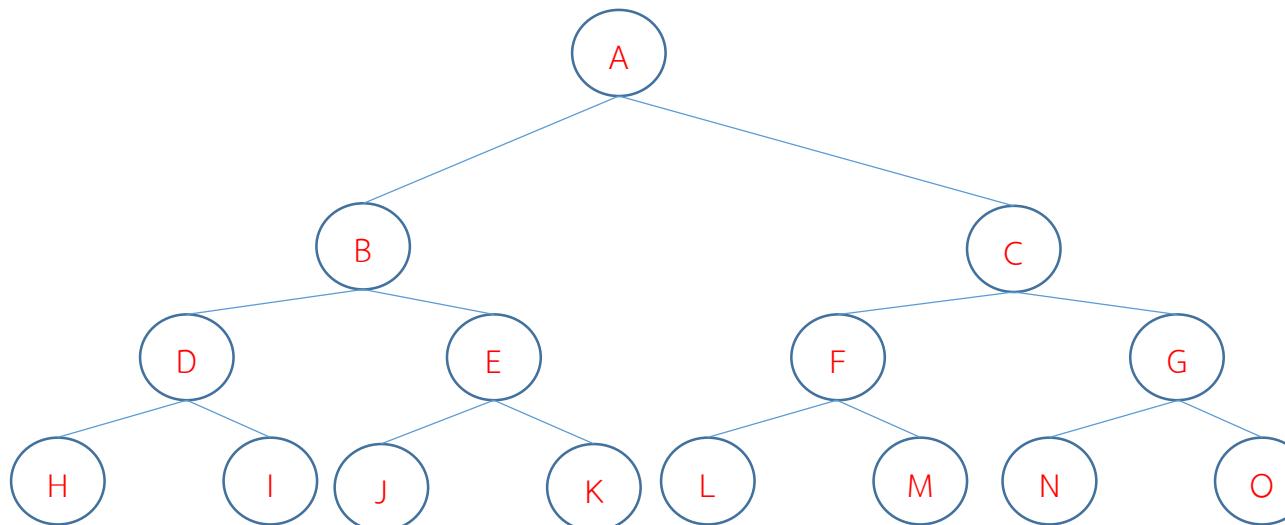
Inorder Traversal

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



การท่องแบบ Inorder

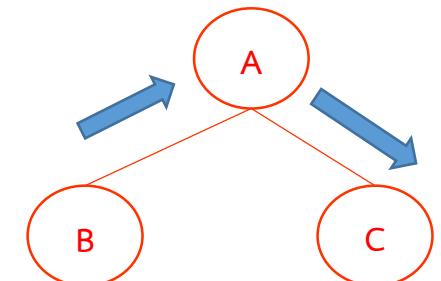
B --> A --> C



Trees: Traversal

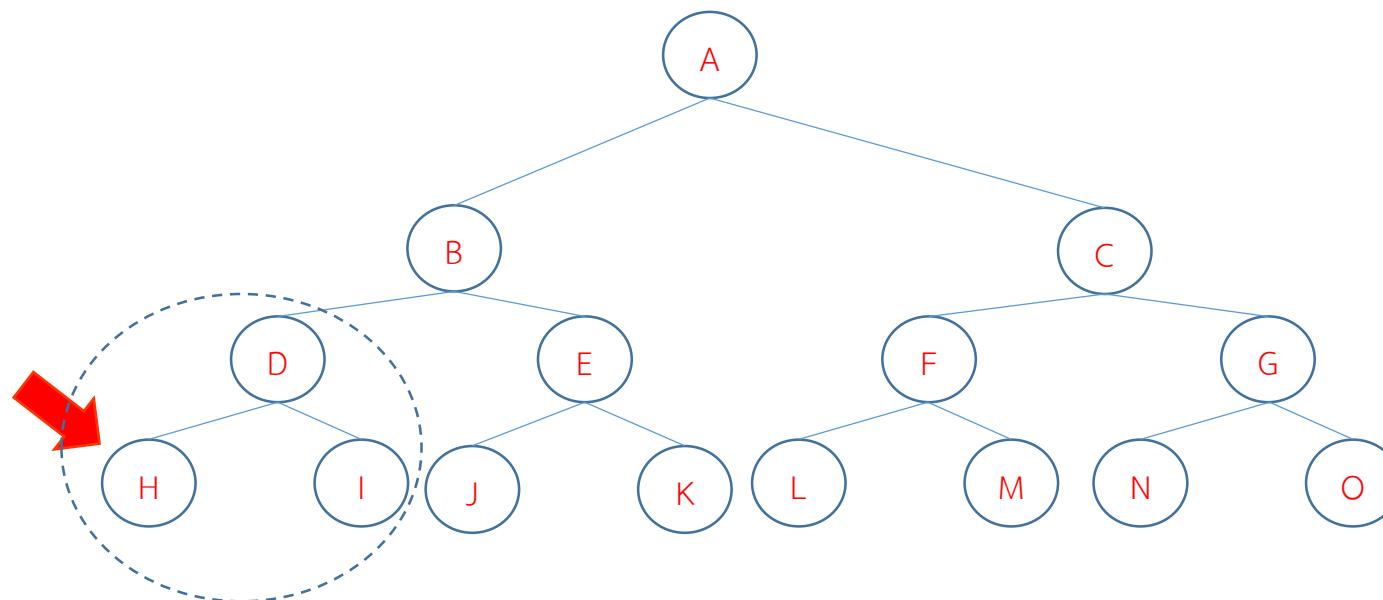
Inorder Traversal

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



การท่องแบบ Inorder

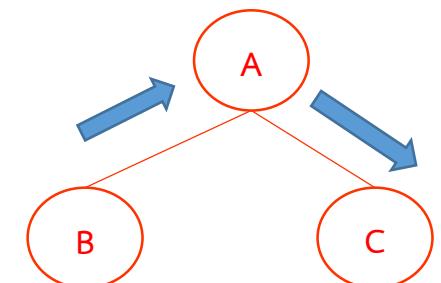
B --> A --> C



Trees: Traversal

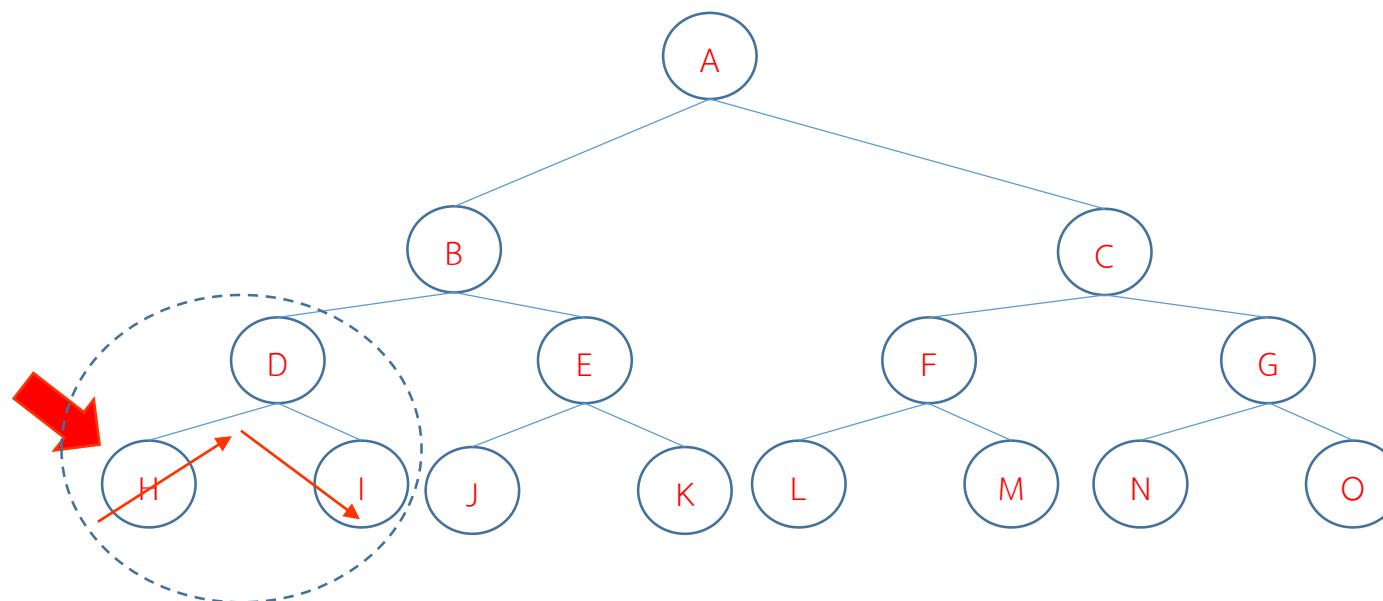
Inorder Traversal

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



การท่องแบบ Inorder

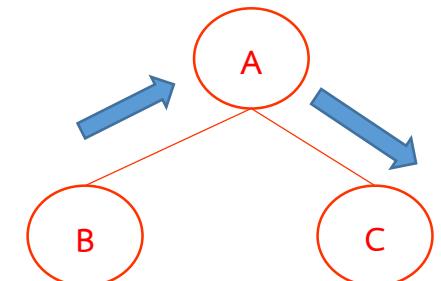
B --> A --> C



Trees: Traversal

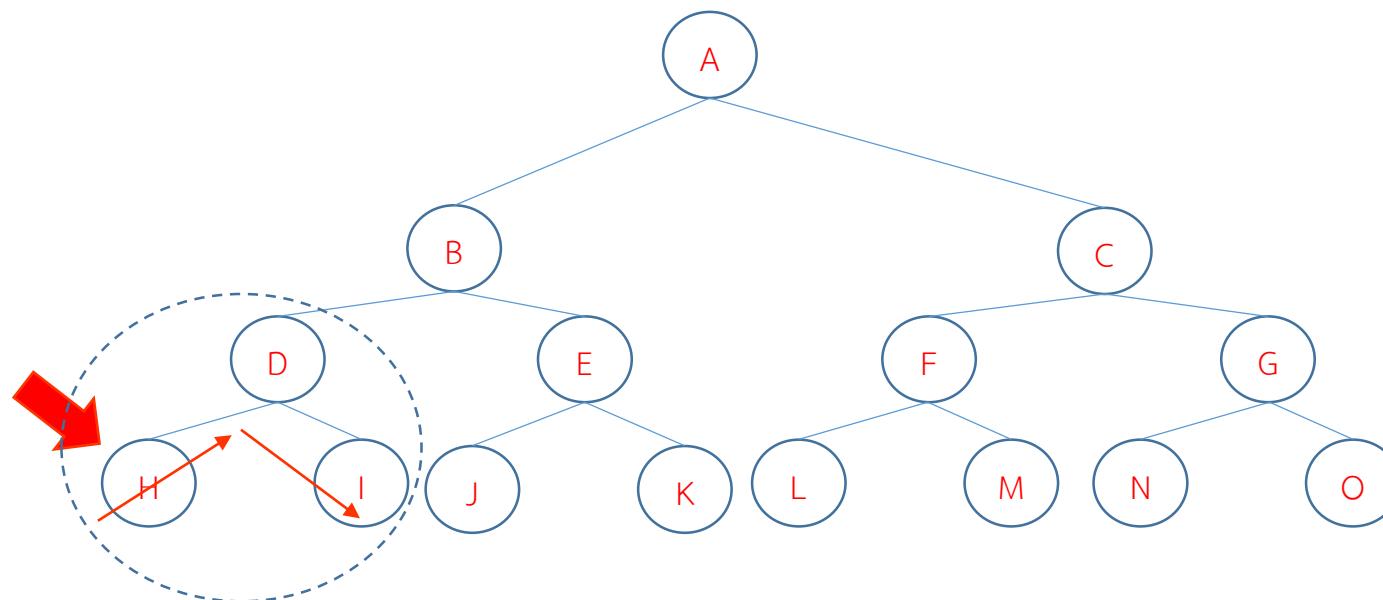
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

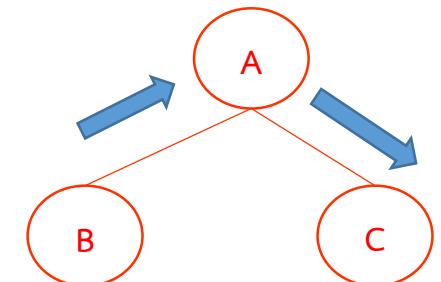


H,D,I

Trees: Traversal

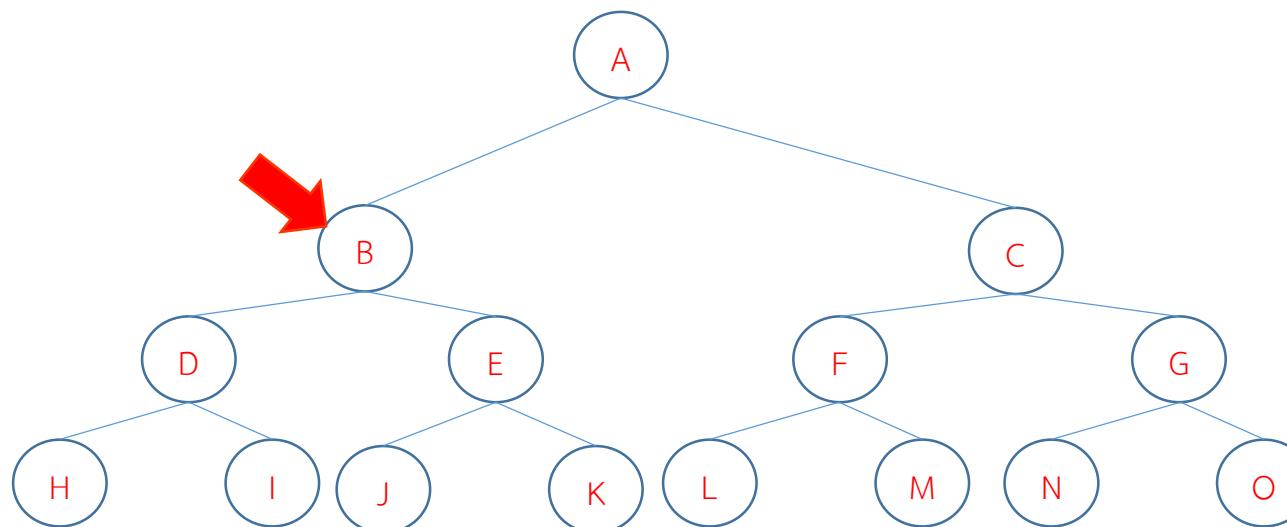
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

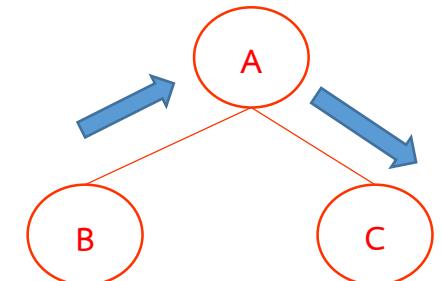


H,D,I,B

Trees: Traversal

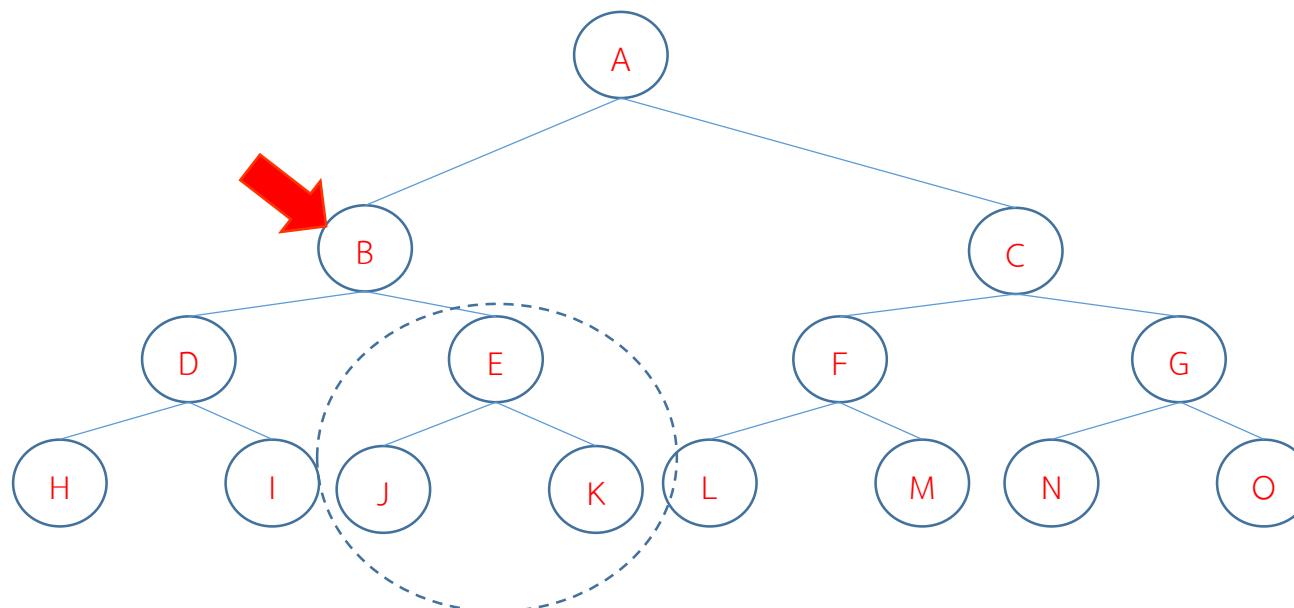
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

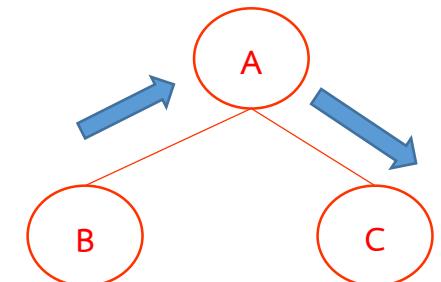


H,D,I,B

Trees: Traversal

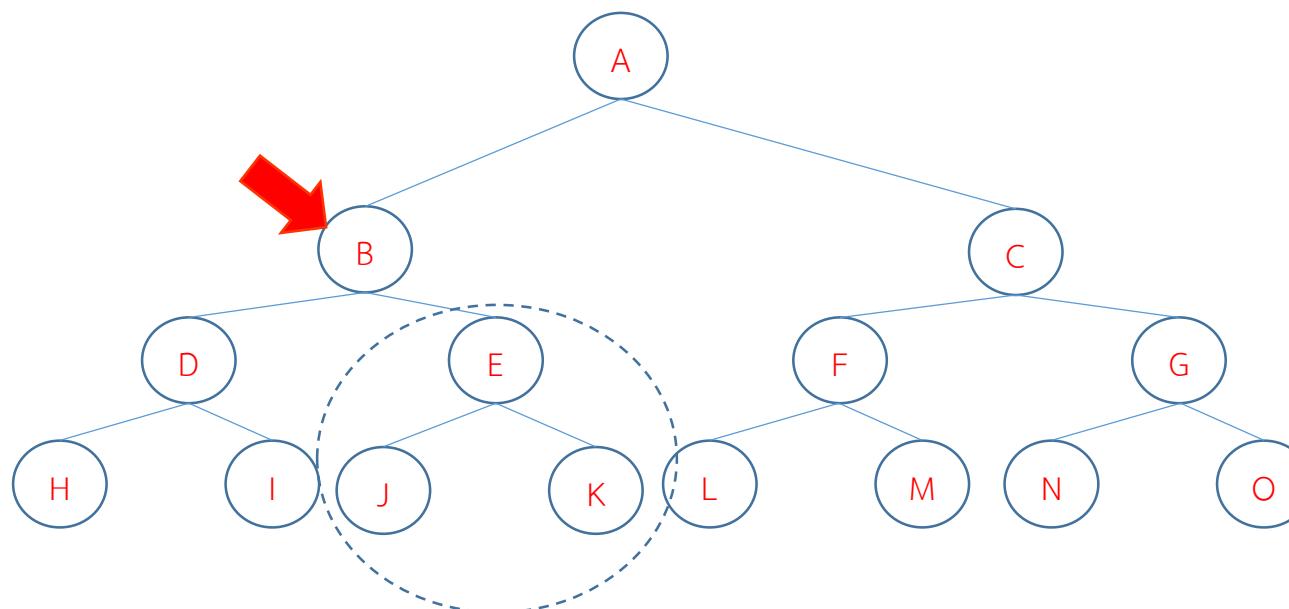
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

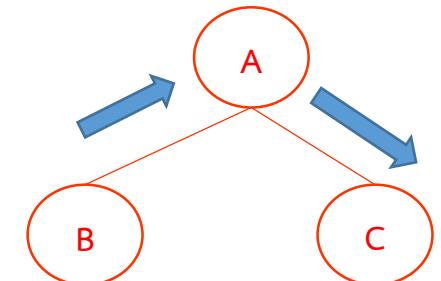


H,D,I,B,J,E,K

Trees: Traversal

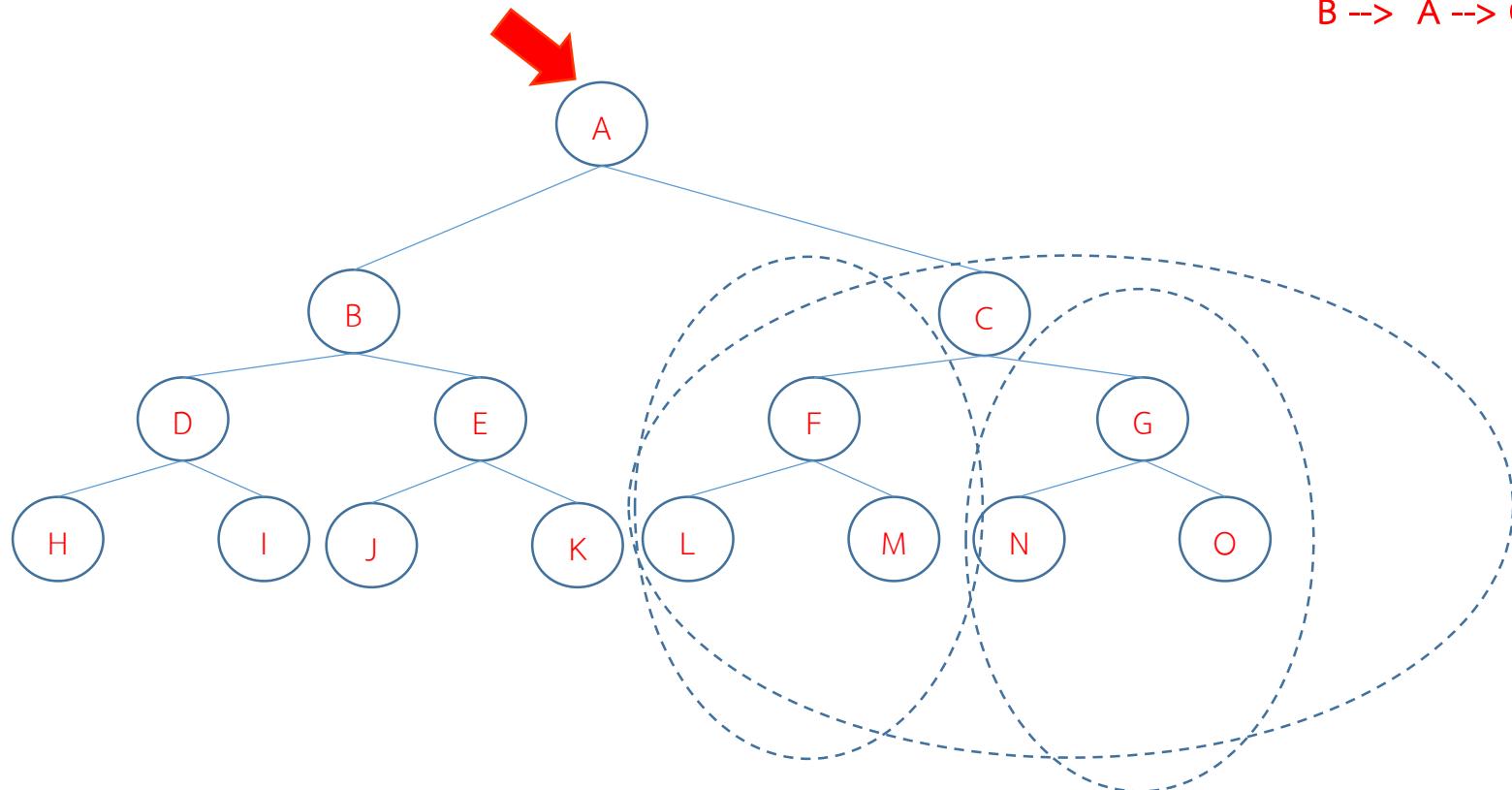
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

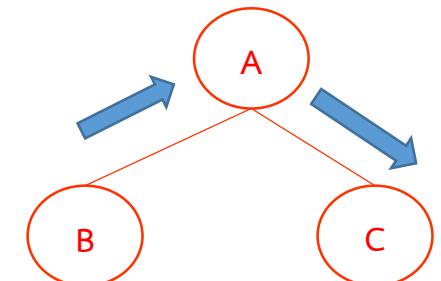


H,D,I,B,J,E,K,A

Trees: Traversal

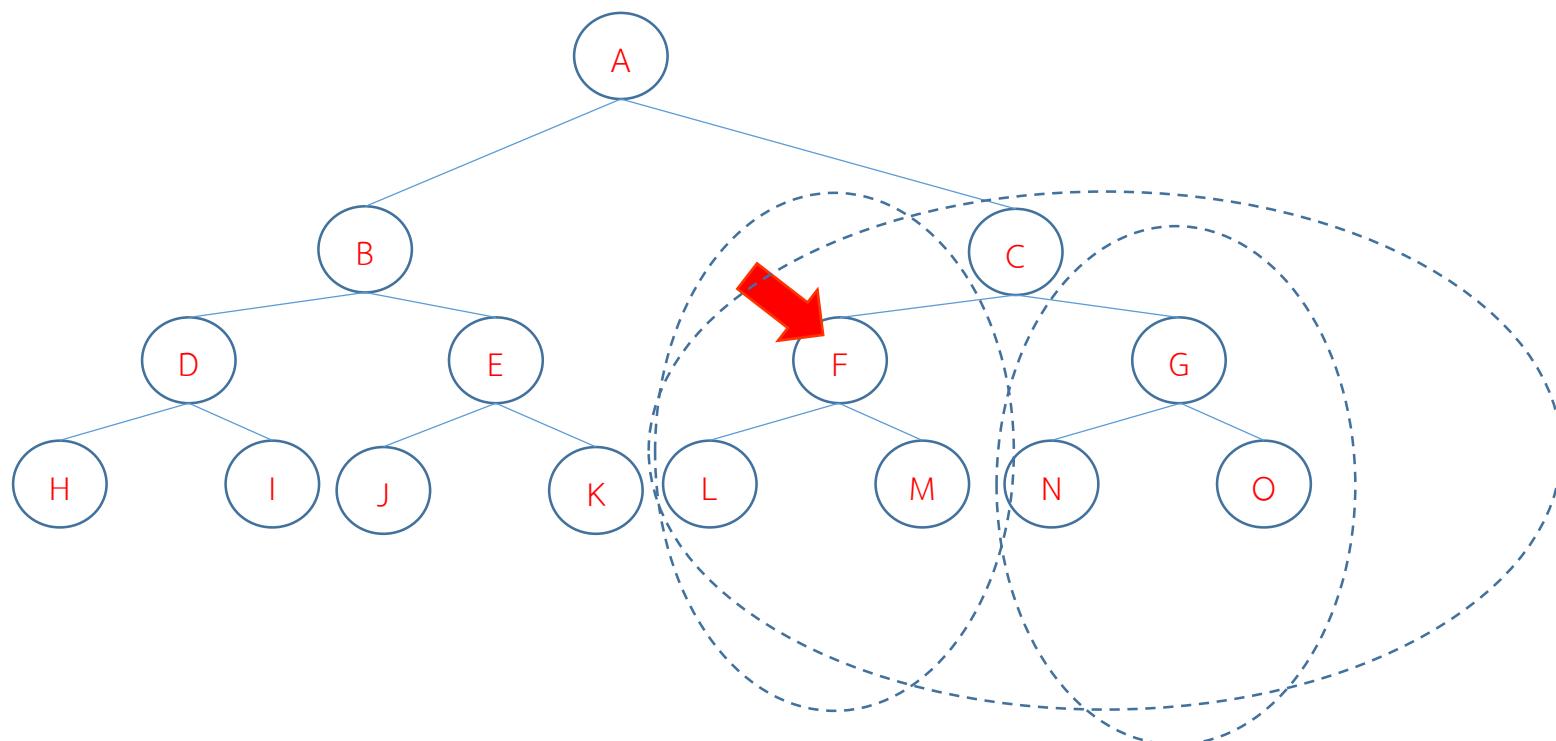
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

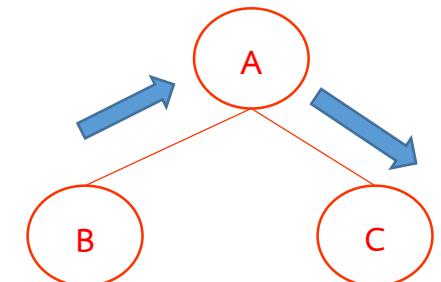


H,D,I,B,J,E,K,A

Trees: Traversal

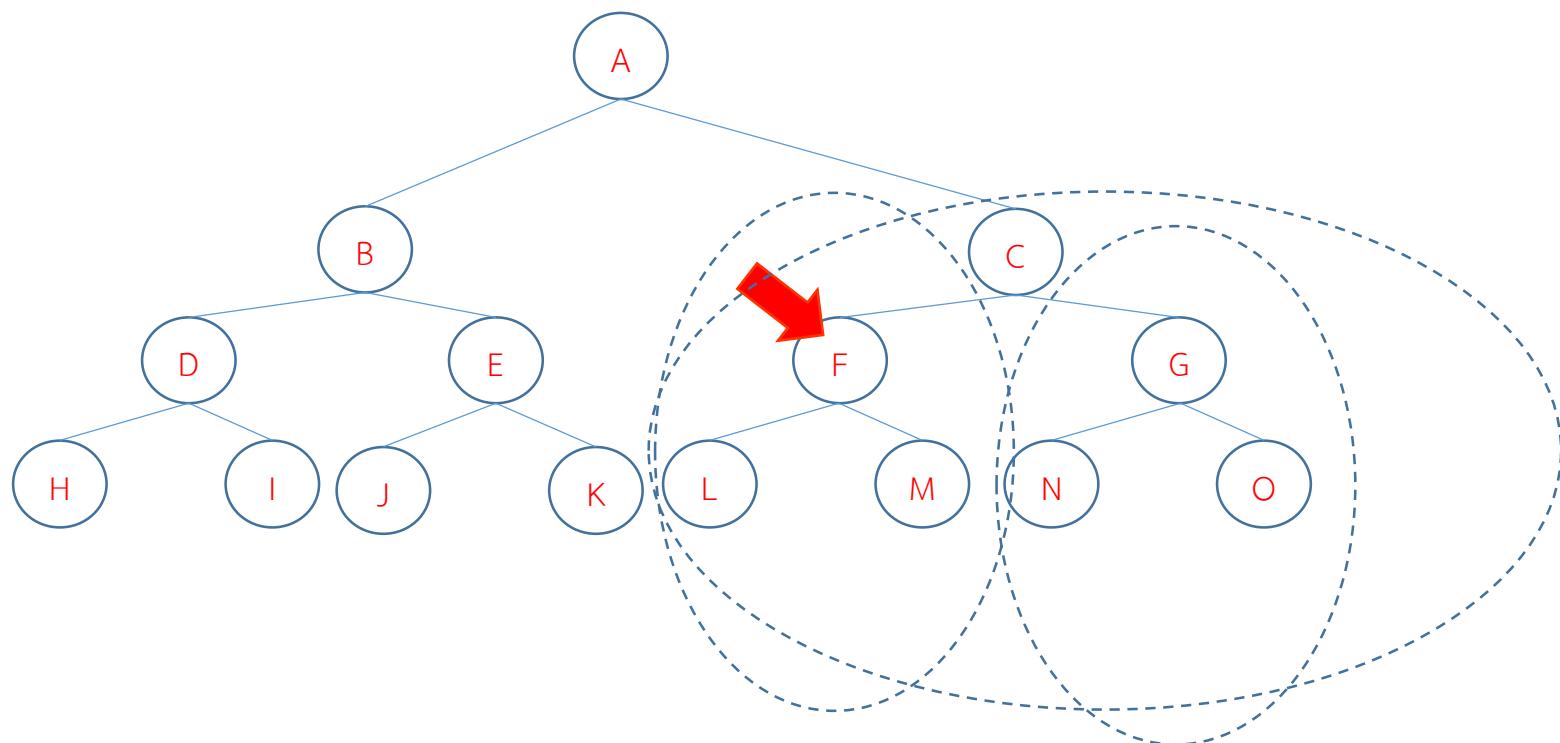
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

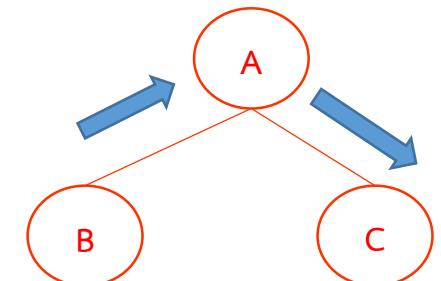


H,D,I,B,J,E,K,A,L,F,M

Trees: Traversal

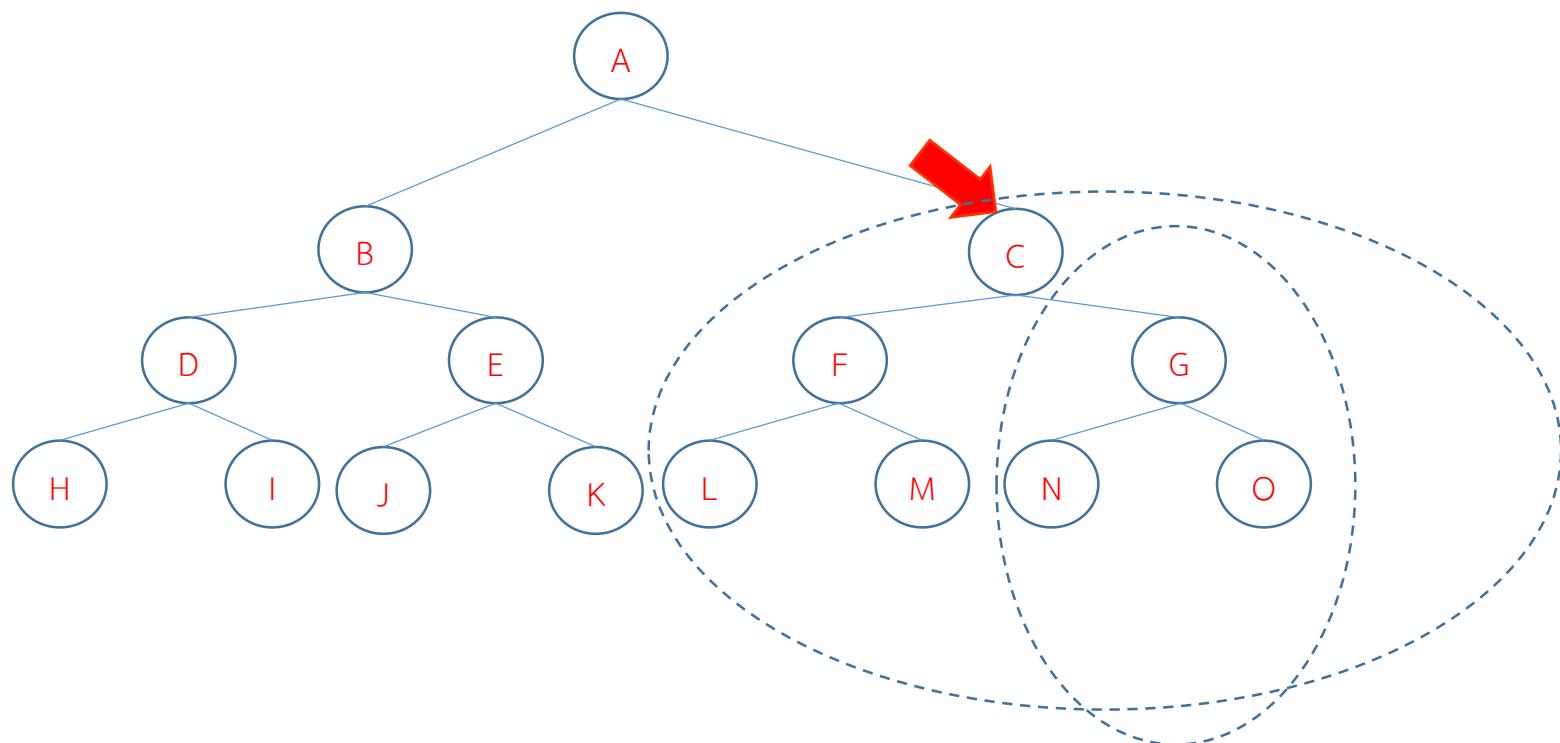
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

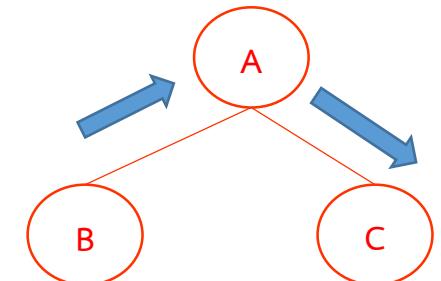


H,D,I,B,J,E,K,A,L,F,M,C

Trees: Traversal

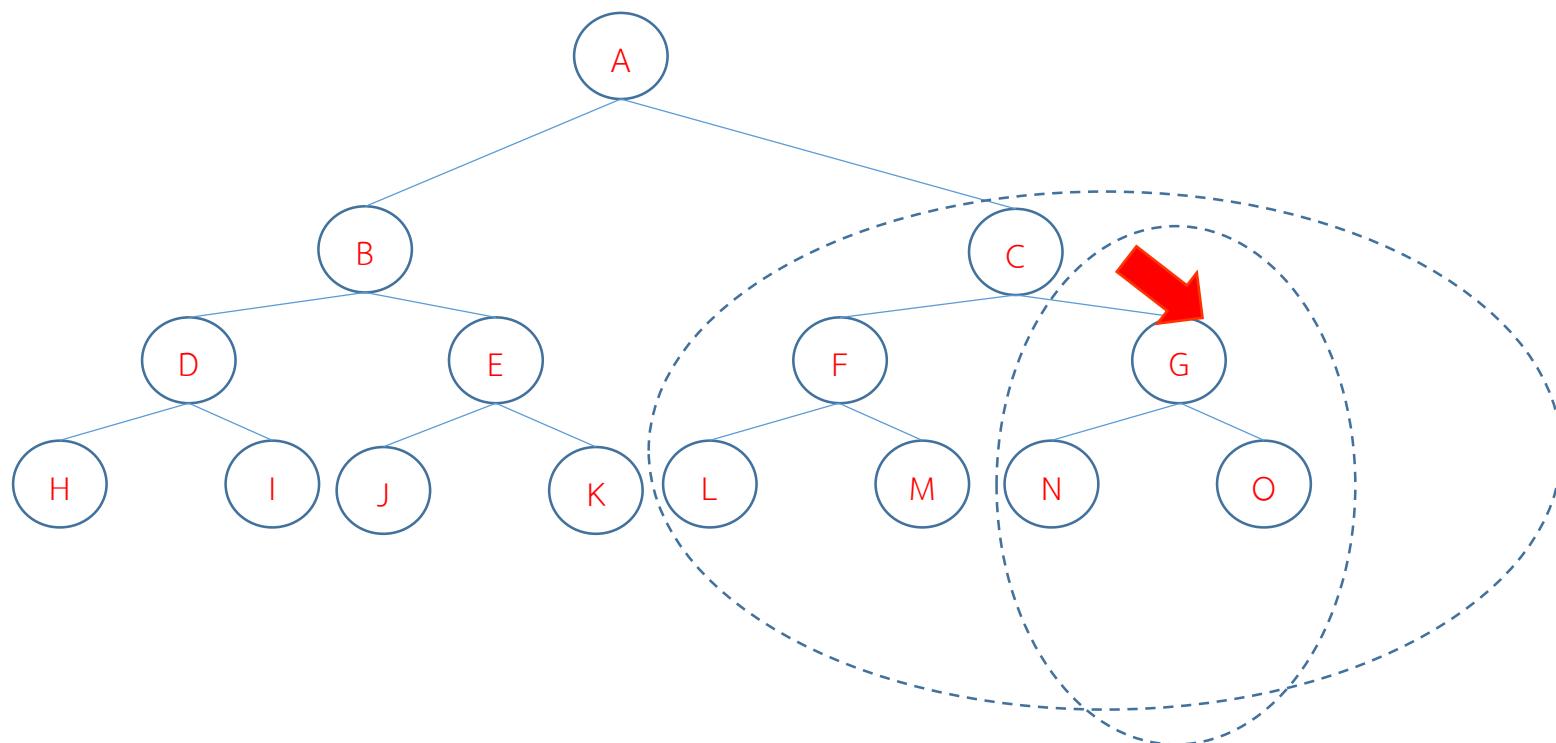
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

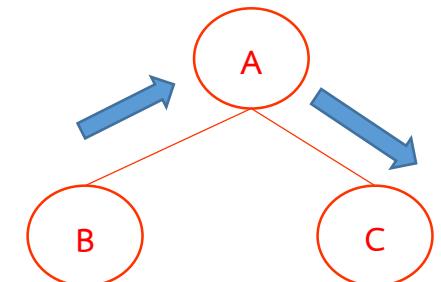


H,D,I,B,J,E,K,A,L,F,M,C

Trees: Traversal

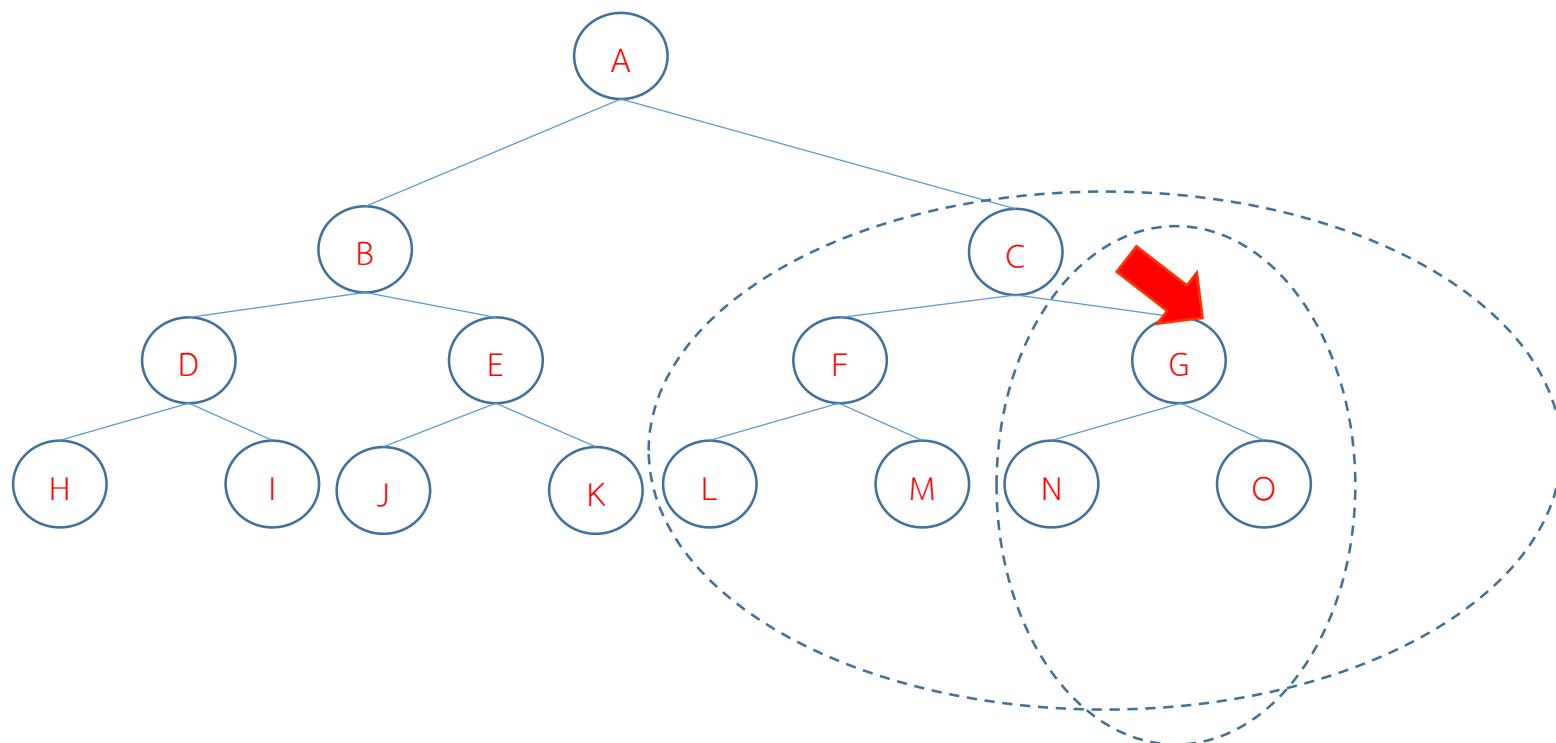
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

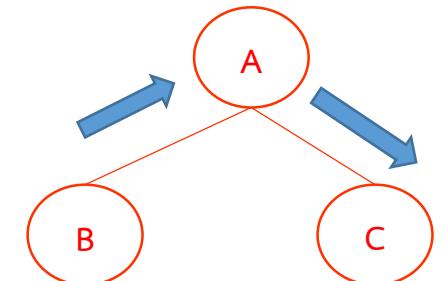


H,D,I,B,J,E,K,A,L,F,M,C,N,G,O

Trees: Traversal

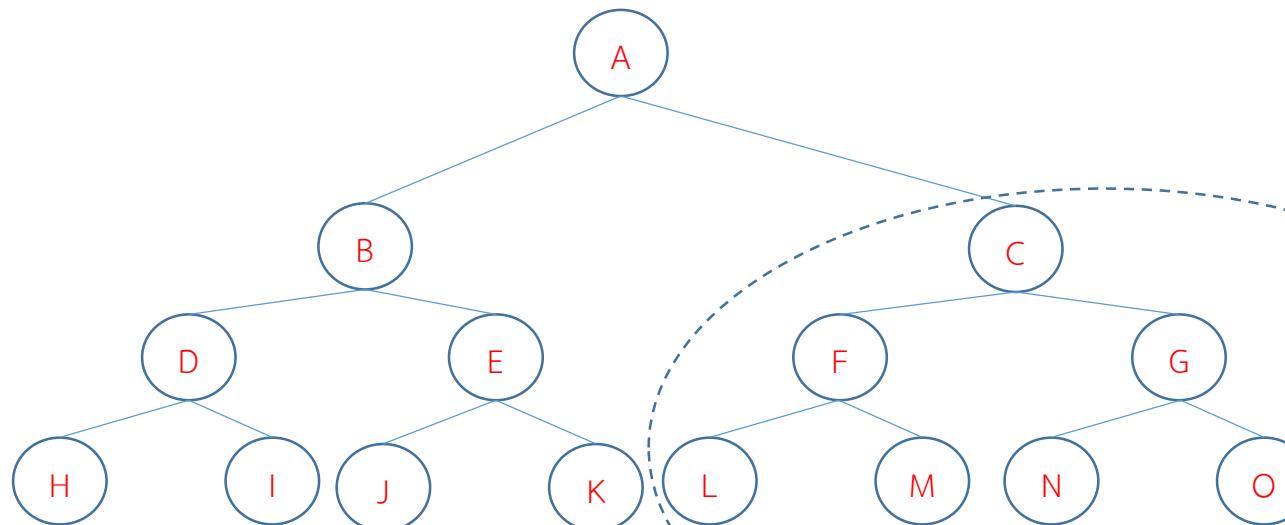
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

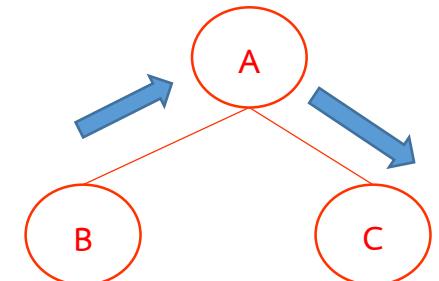


H,D,I,B,J,E,K,A,L,F,M,C,N,G,O

Trees: Traversal

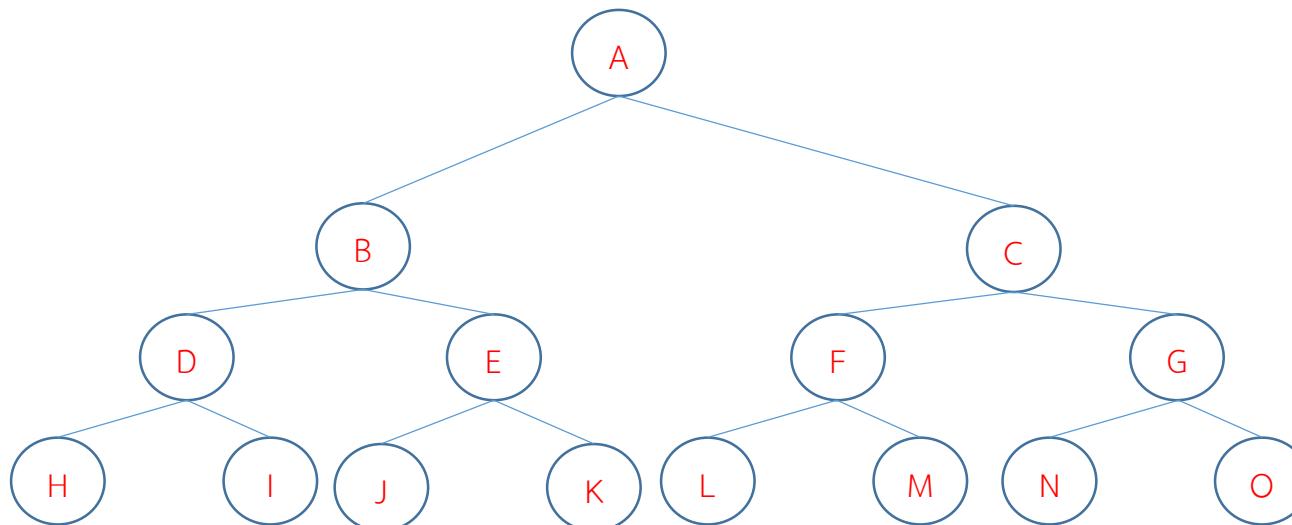
Inorder Traversal

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



การท่องแบบ Inorder

B --> A --> C

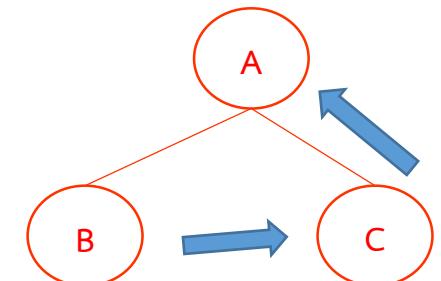


H,D,I,B,J,E,K,A,L,F,M,C,N,G,O

Trees: Traversal

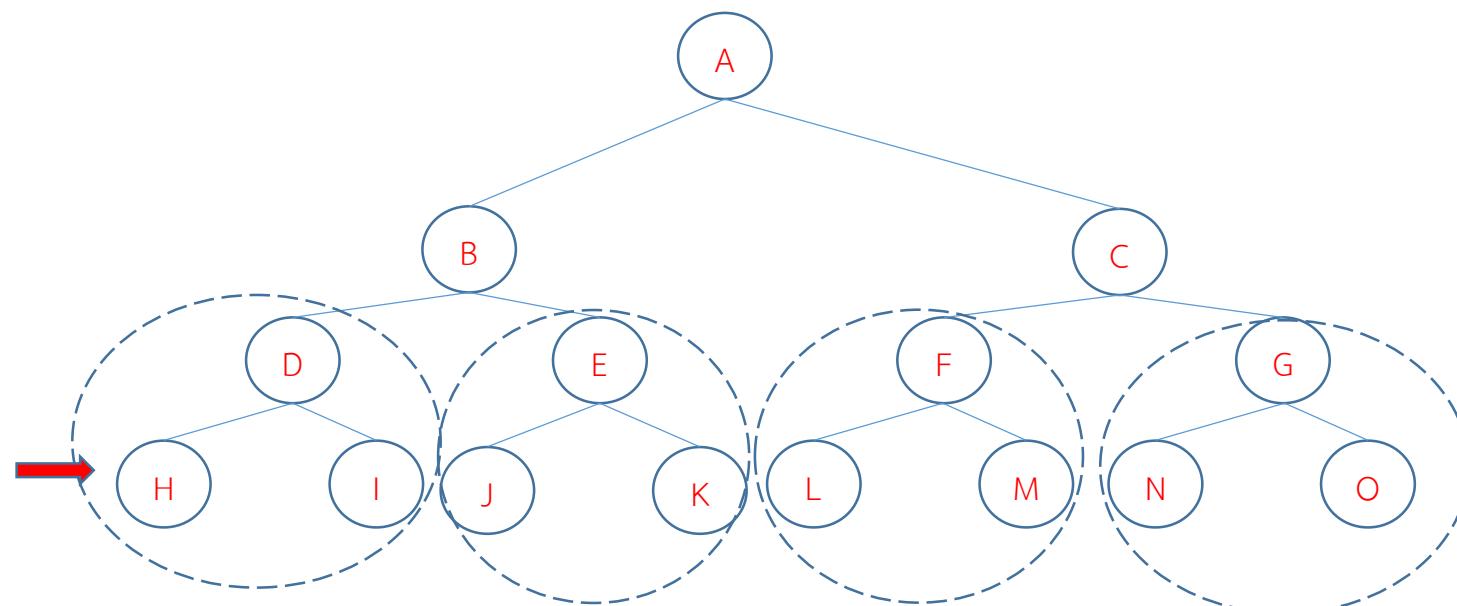
Postorder Traversal

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



การท่องแบบ Inorder

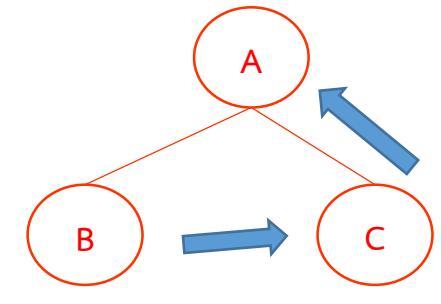
B --> C --> A



Trees: Traversal

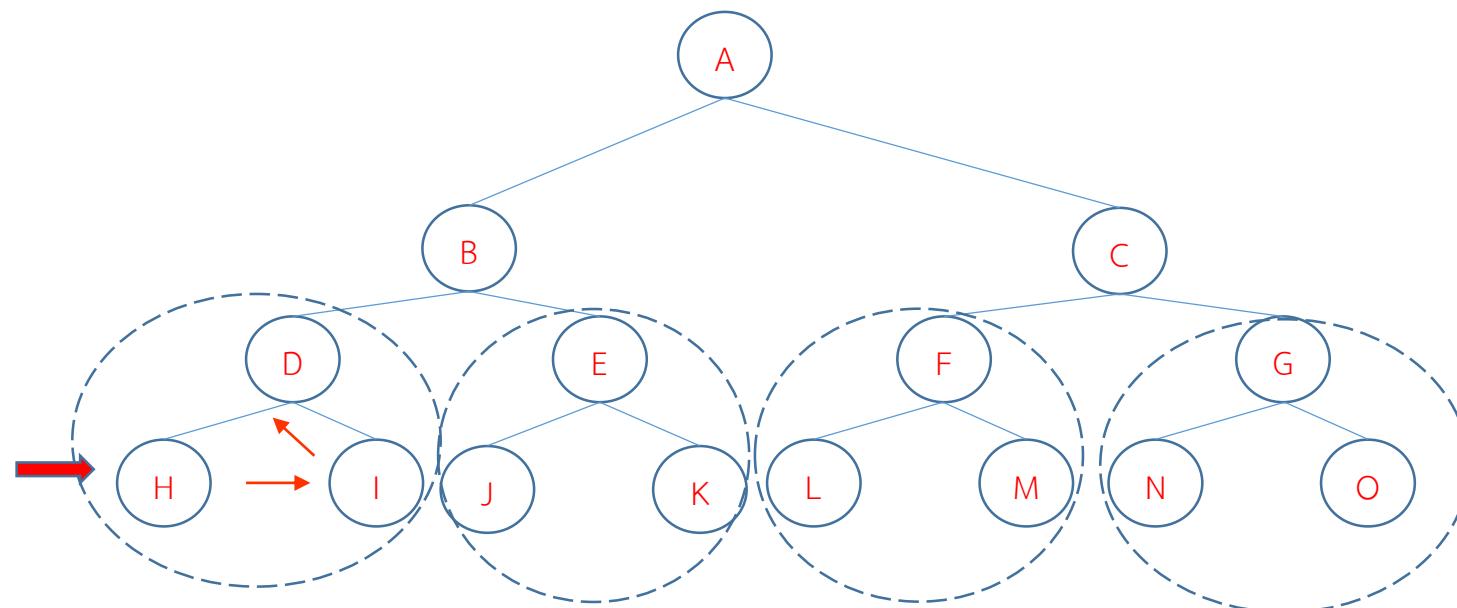
Postorder Traversal

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



การท่องแบบ Inorder

B --> C --> A

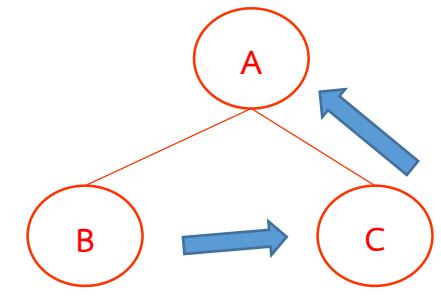


H,I,D

Trees: Traversal

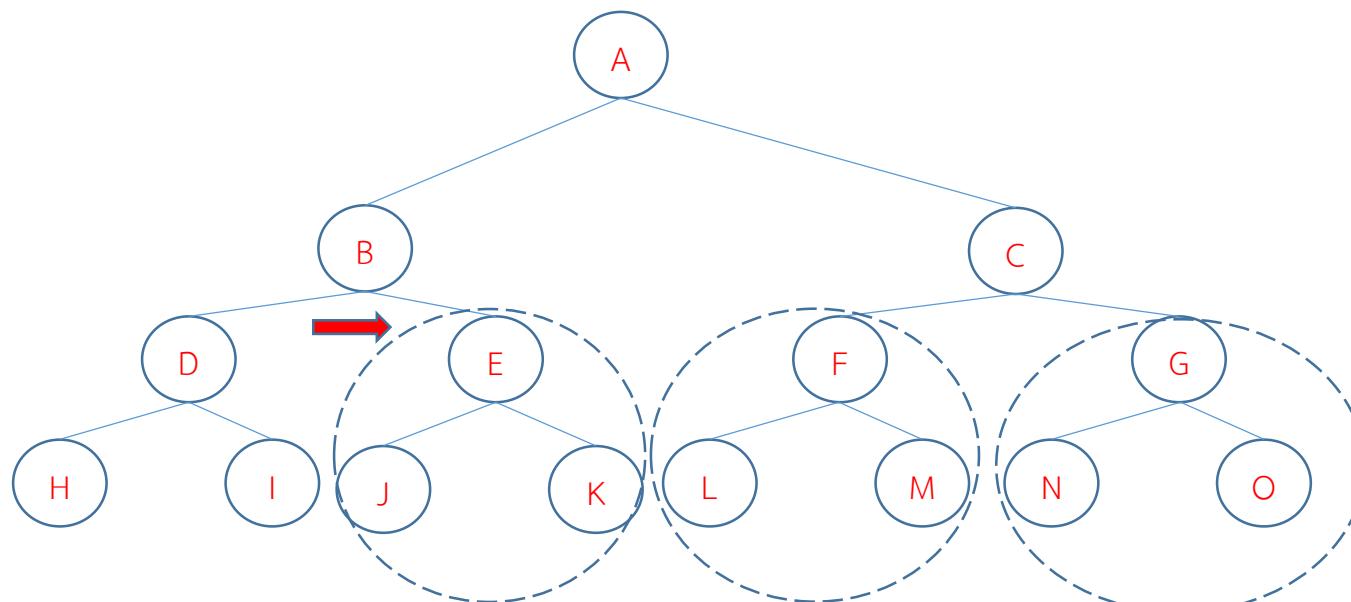
Postorder Traversal

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



การท่องแบบ Inorder

B --> C --> A

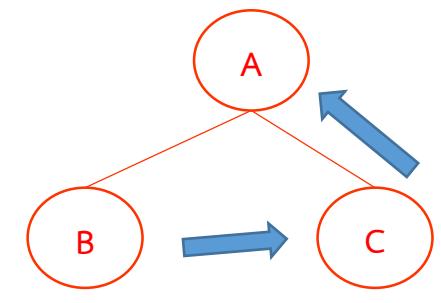


H,I,D

Trees: Traversal

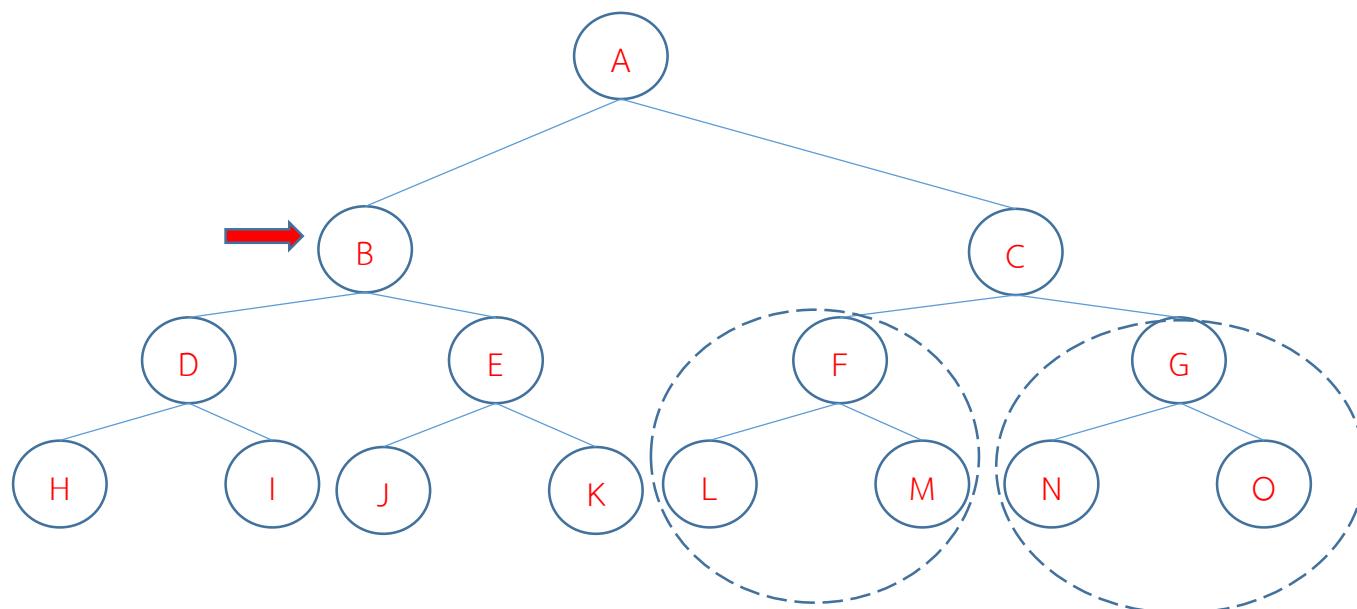
Postorder Traversal

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



การท่องแบบ Inorder

B --> C --> A

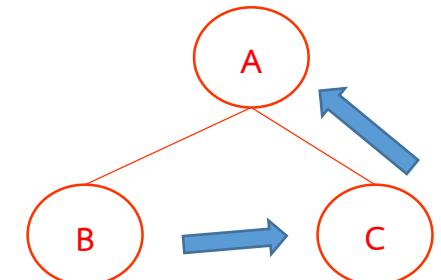


H,I,D,J,K,E,B

Trees: Traversal

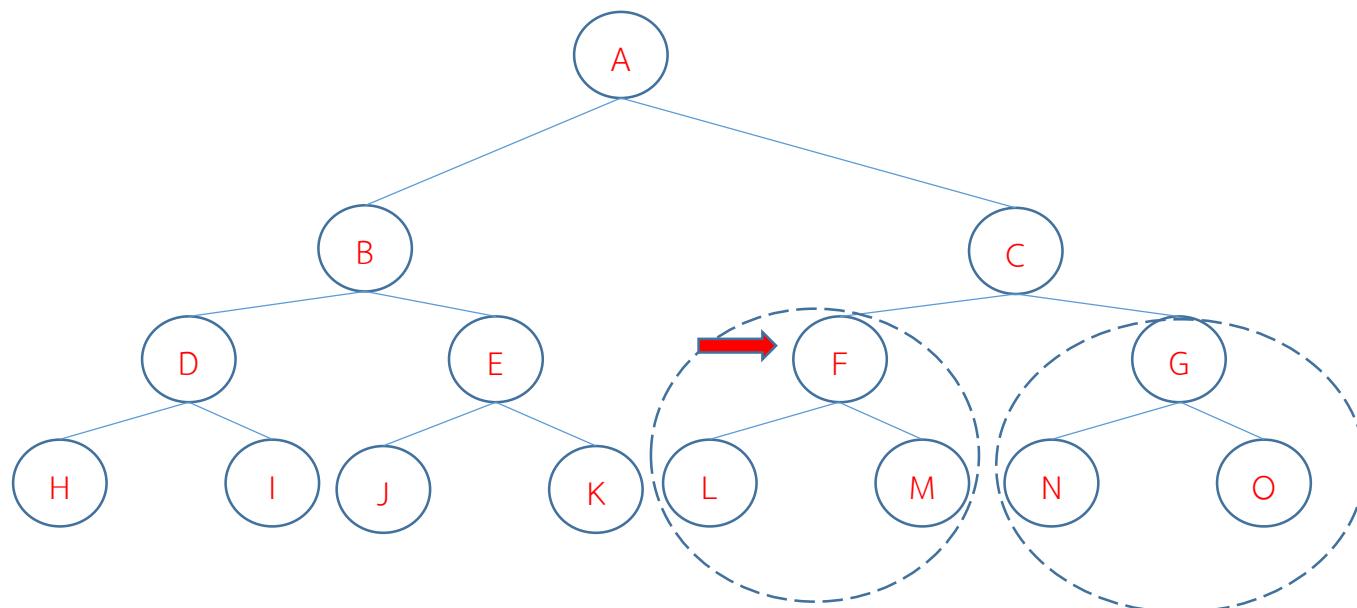
Postorder Traversal

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



การท่องแบบ Inorder

B --> C --> A

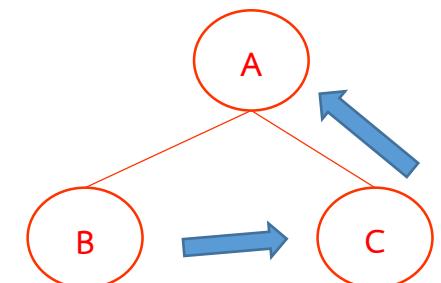


H,I,D,J,K,E,B,L,M,F

Trees: Traversal

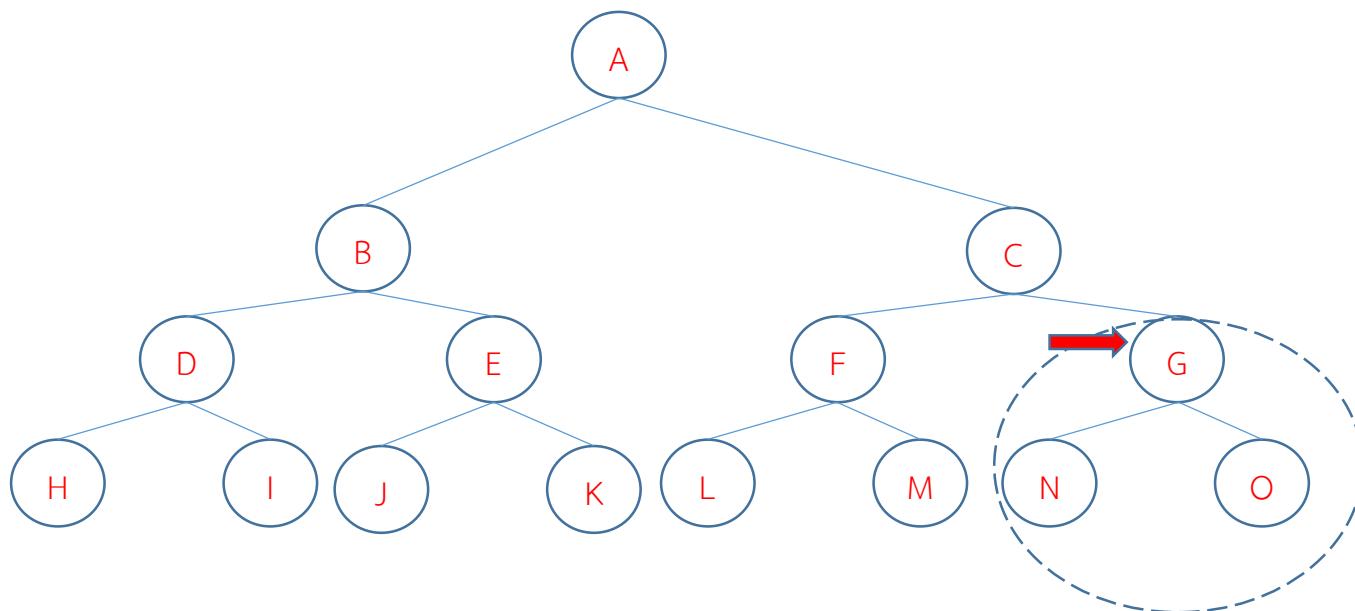
Postorder Traversal

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



การท่องแบบ Inorder

B --> C --> A

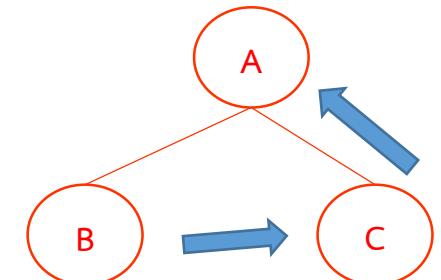


H,I,D,J,K,E,B,L,M,F,N,O,G

Trees: Traversal

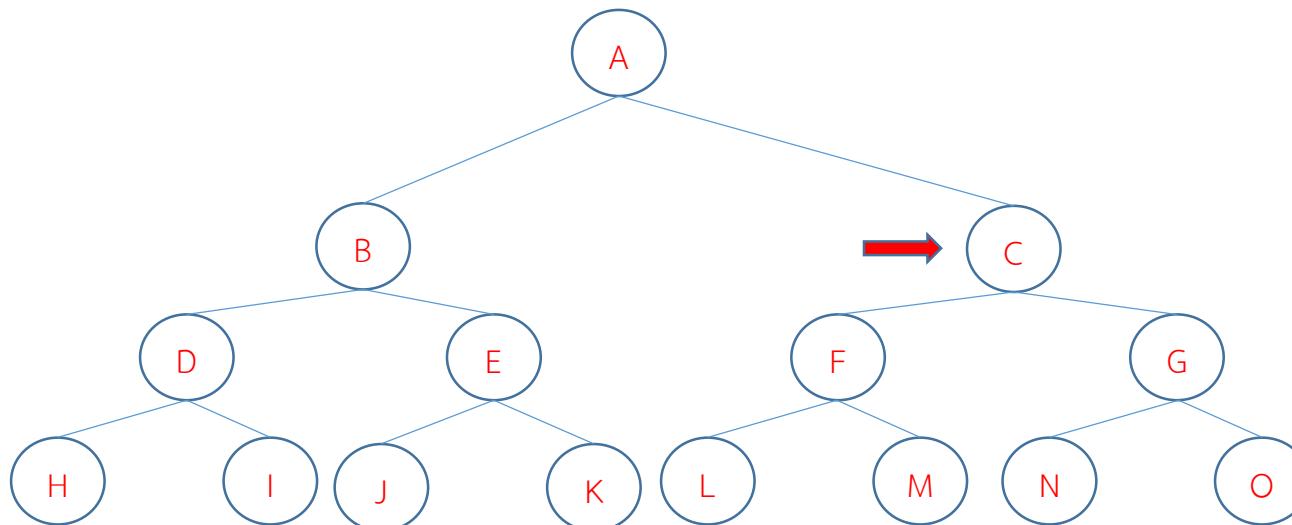
Postorder Traversal

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



การท่องแบบ Inorder

B --> C --> A

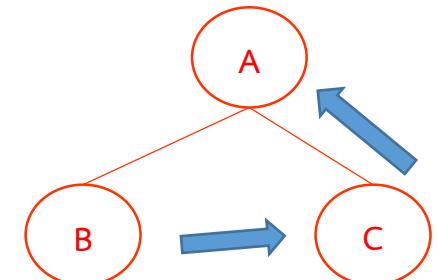


H,I,D,J,K,E,B,L,M,F,N,O,G,C

Trees: Traversal

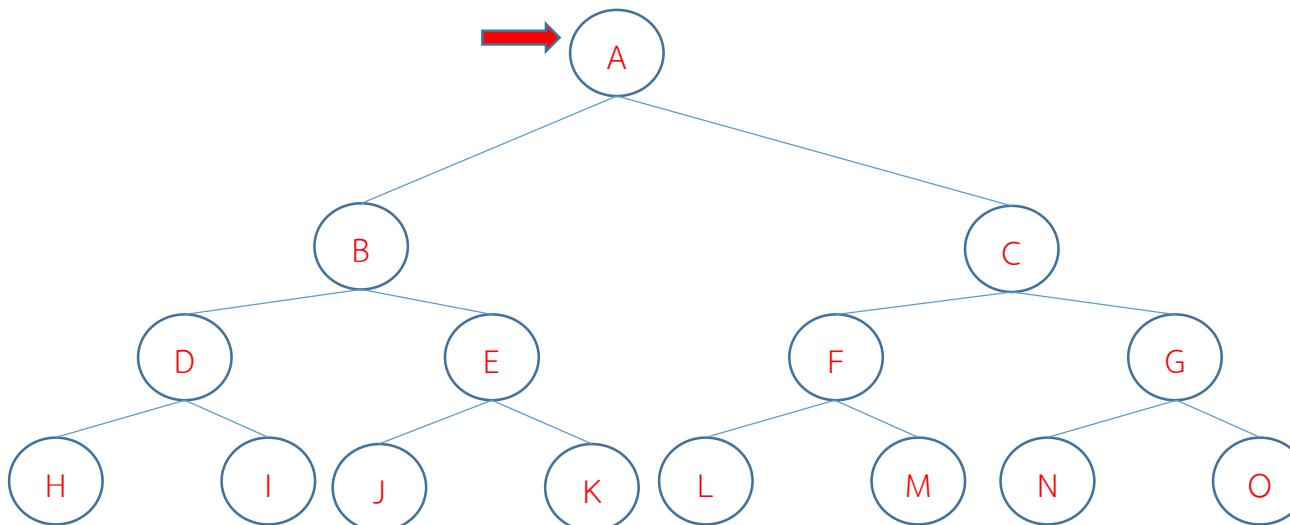
Postorder Traversal

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



การท่องแบบ Inorder

B --> C --> A

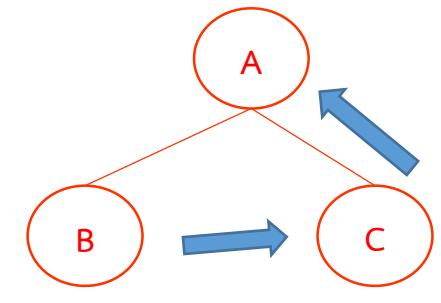


H,I,D,J,K,E,B,L,M,F,N,O,G,C,A

Trees: Traversal

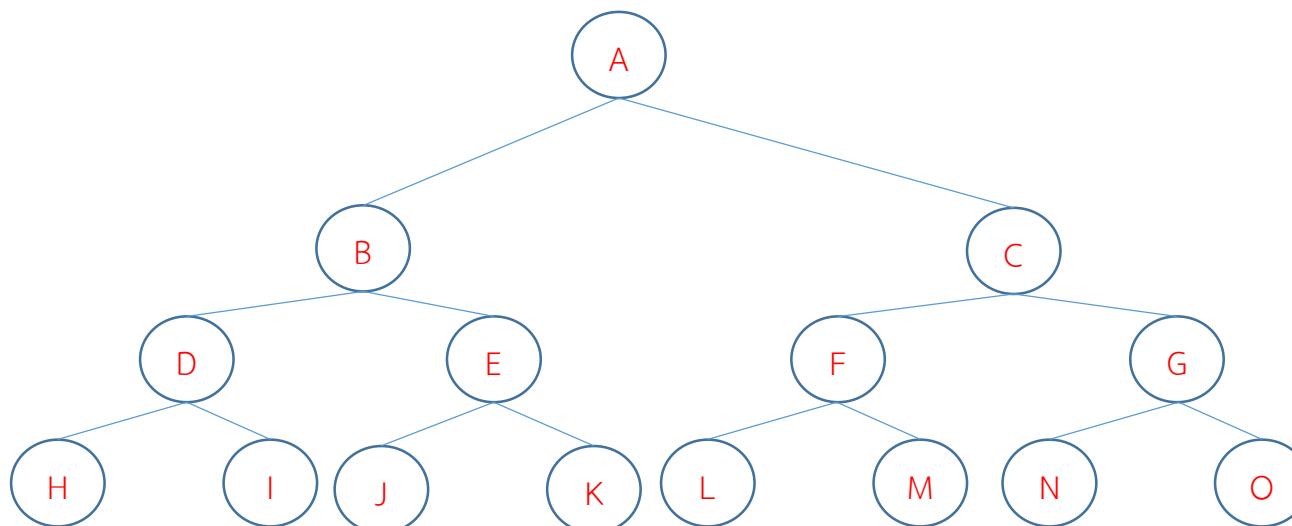
Postorder Traversal

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



การท่องแบบ Inorder

B --> C --> A



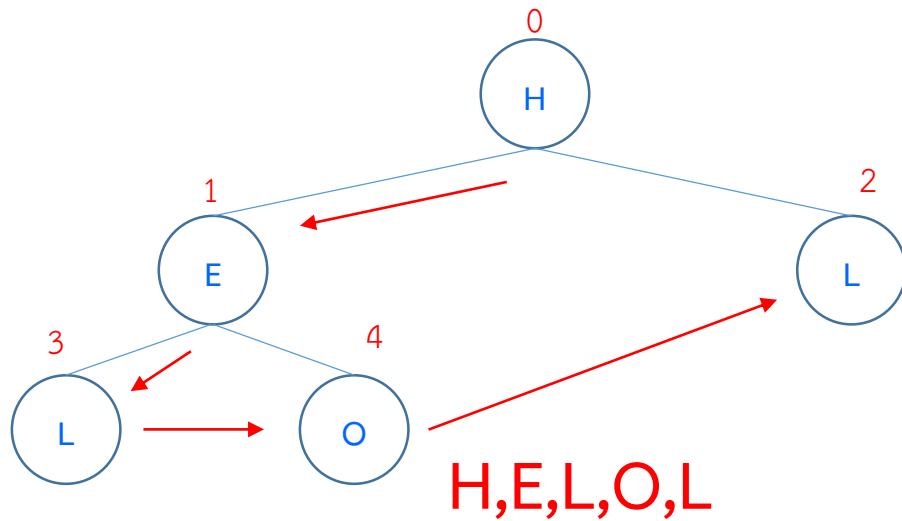
H,I,D,J,K,E,B,L,M,F,N,O,G,C,A

ขั้นตอนวิธีการท่องไปในต้นไม้

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

Trees: Preorder Traverse Algorithm

การท่องไปในต้นไม้แบบ Preorder แบบ Recursive



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

```
#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]);
}
}

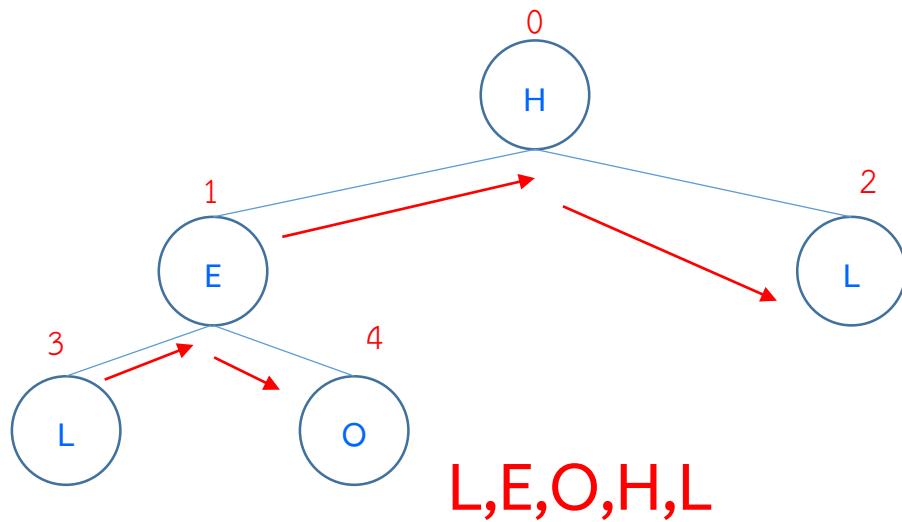
void preorder(void){
    pre(0);
}

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

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

Trees: Inorder Traverse Algorithm

การท่องไปในต้นไม้แบบ Inorder แบบ Recursive



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

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

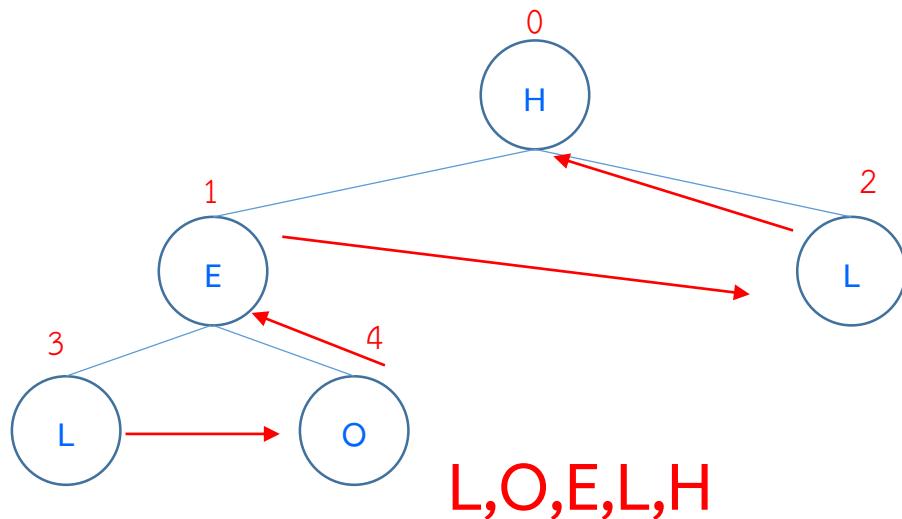
void inord(p){
    if(p!=-1){
        inord(Left[p]);
        printf("%c ",Data[p]);
        inord(Right[p]);
    }
}
void inorder()
{
    inord(0);
}

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

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

Trees: Postorder Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบ Recursive



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

```
#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 . . .

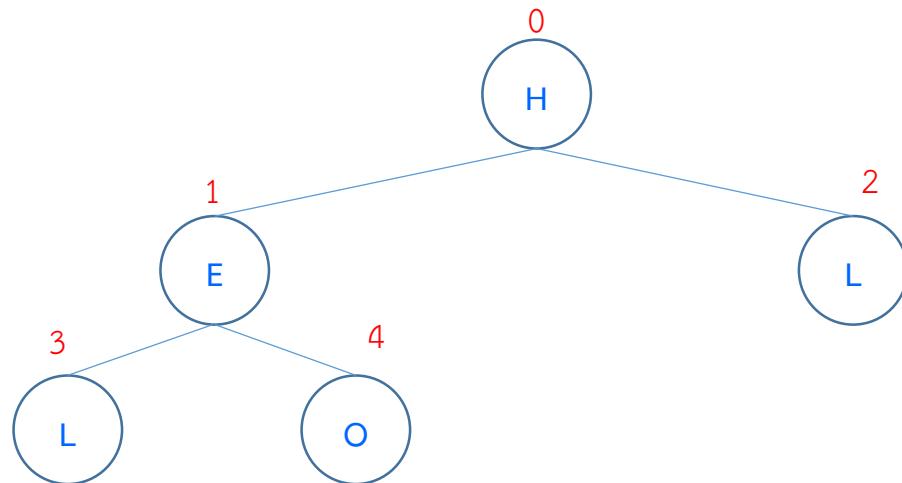
การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

- 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

Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

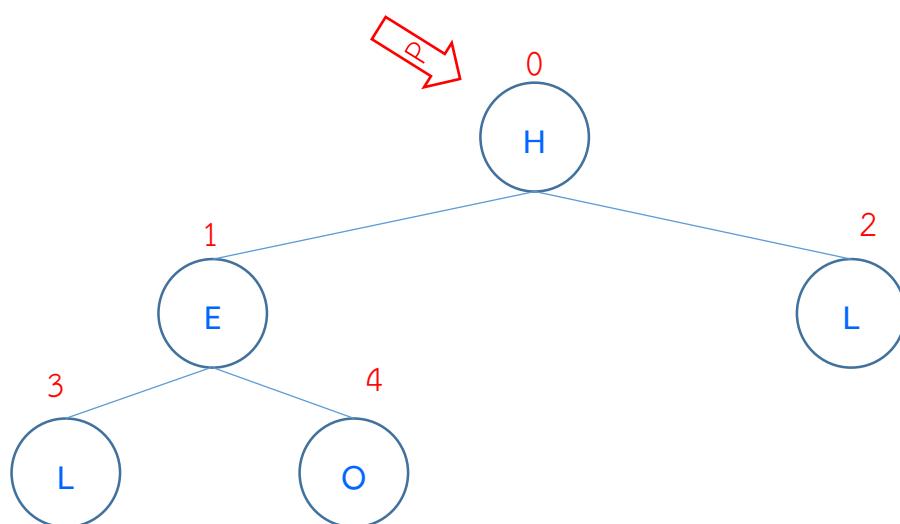
- 1) push ตำแหน่งของ root node
 - 2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
 - 3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
 - 4) pop มาค่าใส่ p



Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

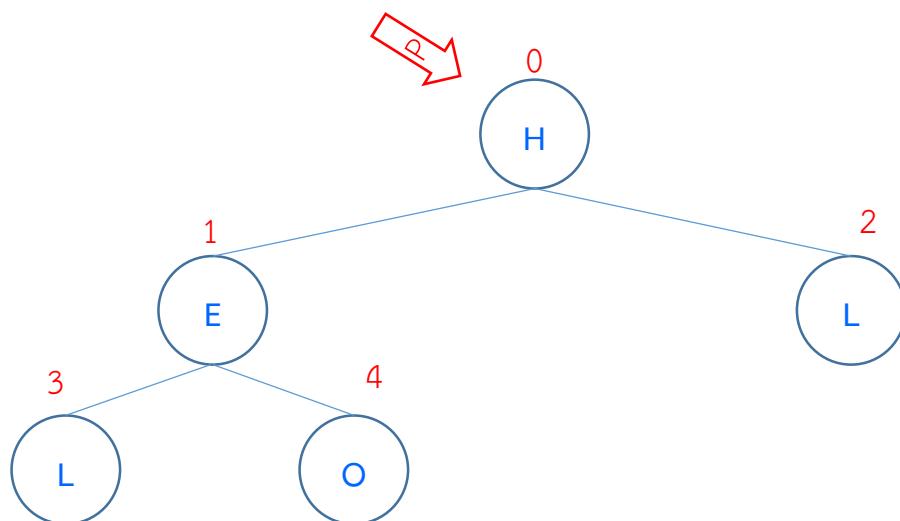
- 1) push ตำแหน่งของ root node
 - 2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
 - 3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
 - 4) pop มาค่าใส่ p



Trees: Preorder Traverse Algorithm

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

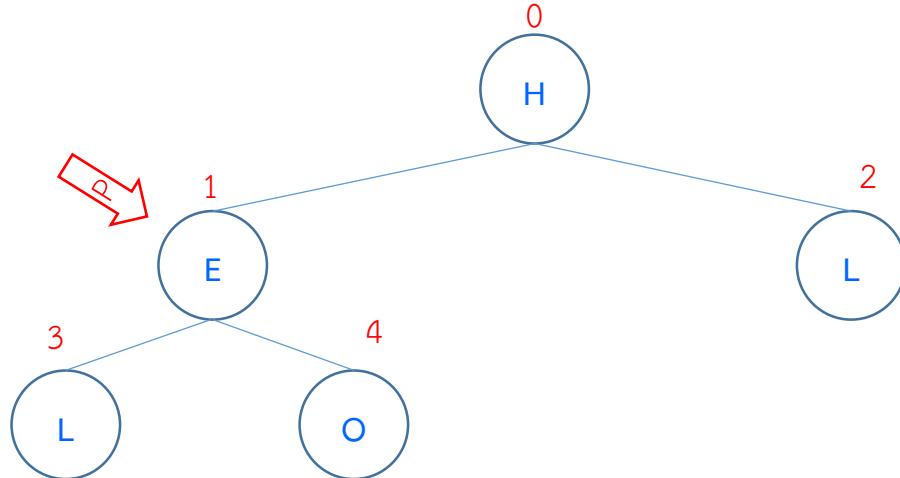
- 1) push ตำแหน่งของ root node
 - 2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
 - 3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
 - 4) pop มาค่าใส่ p



Trees: Preorder Traverse Algorithm

การท่องไปในต้นไม้แบบ Preorder แบบใช้ Stack

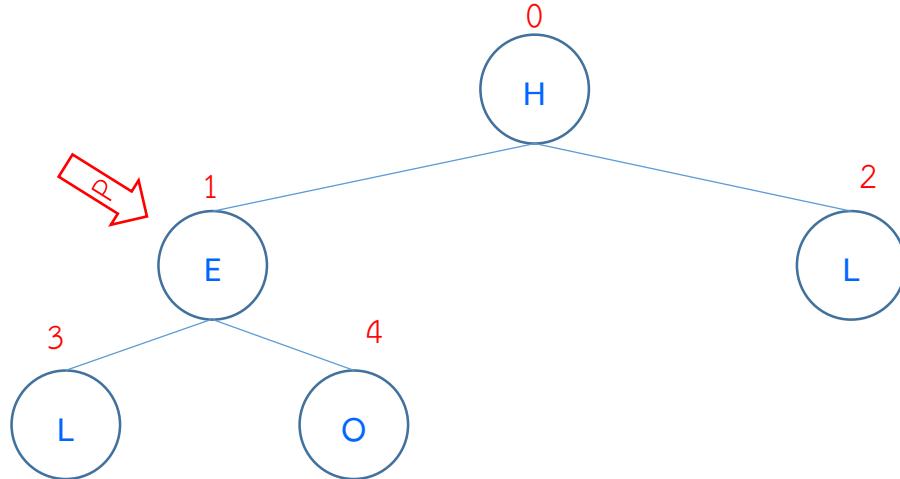
- 1) push ตำแหน่งของ root node
 - 2) กำหนดให้ p คือตำแหน่งของ root node และทำข้อ 3 จนกว่า p จะเป็น null
 - 3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p ถ้ามี node ด้านขวาให้ push address ของมัน โดยให้หยุดทำขั้นตอน 3 เมื่อไปถึง leaf node
 - 4) pop มาค่าใส่ p



Trees: Preorder Traverse Algorithm

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

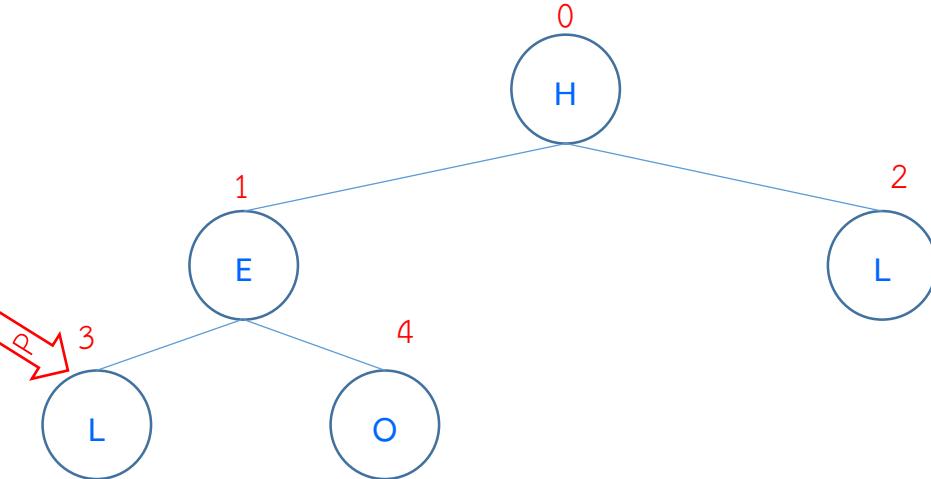


p	Stack	Node ที่ผ่าน
	0	
0	ว่า	H
0	2	H
1	2	H E
1	4 2	H E

Trees: Preorder Traverse Algorithm

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

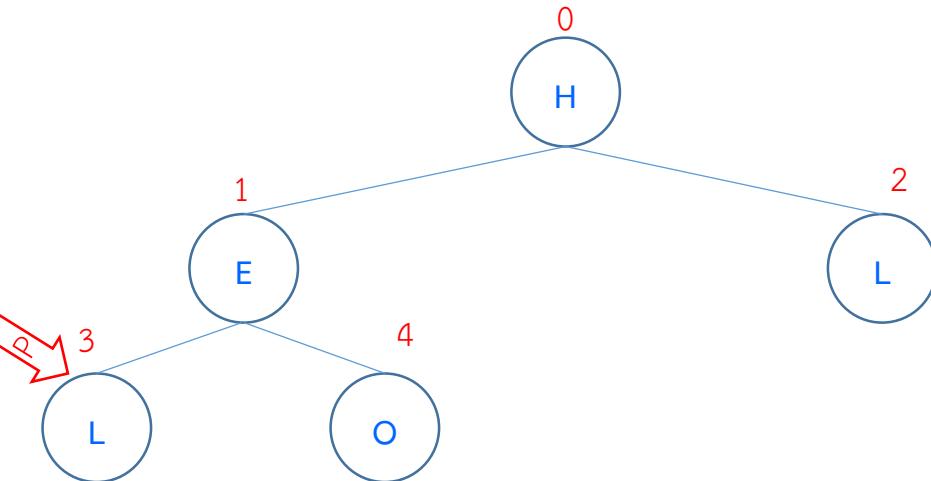


p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L

Trees: Preorder Traverse Algorithm

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

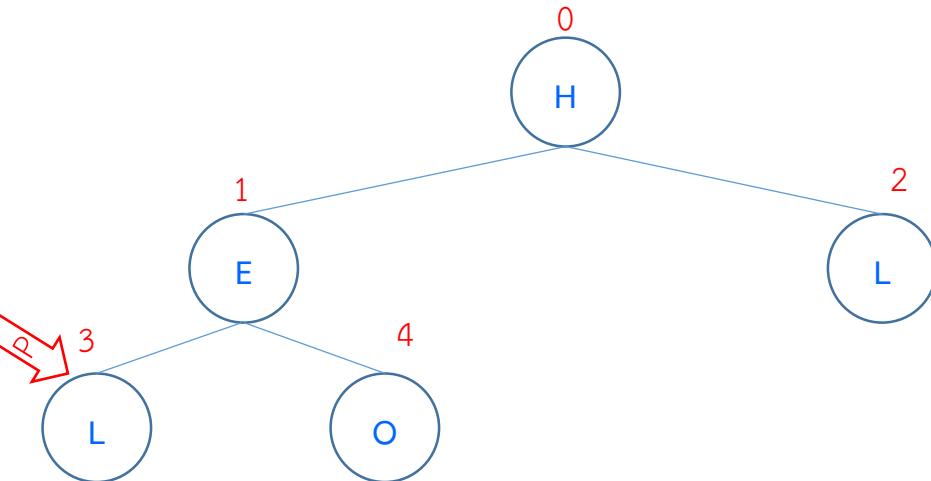


p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L

Trees: Preorder Traverse Algorithm

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

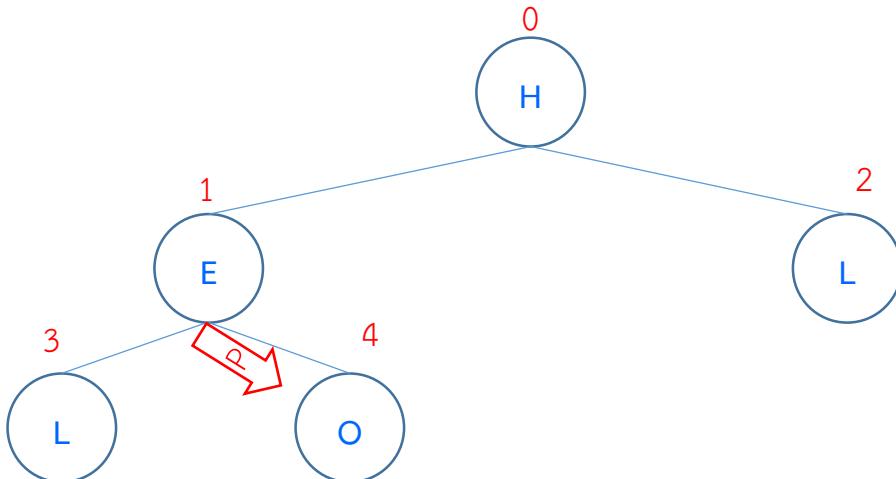


p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L

Trees: Preorder Traverse Algorithm

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

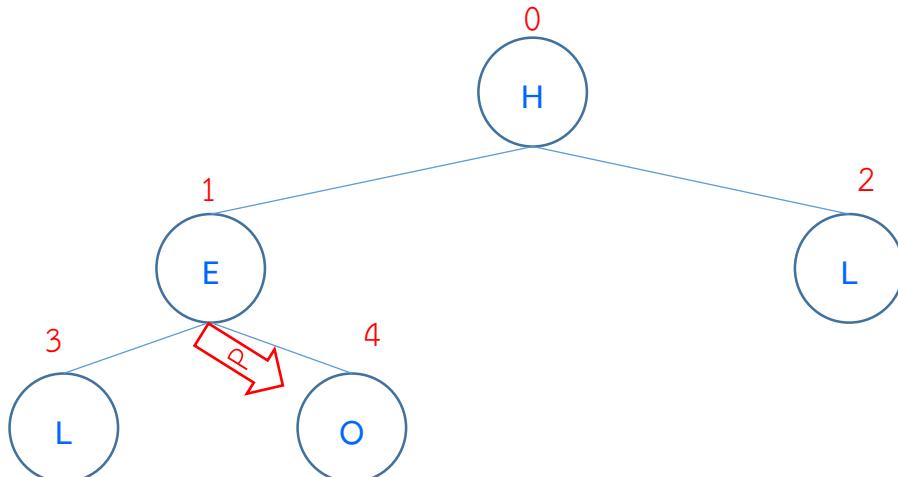


p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L O

Trees: Preorder Traverse Algorithm

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

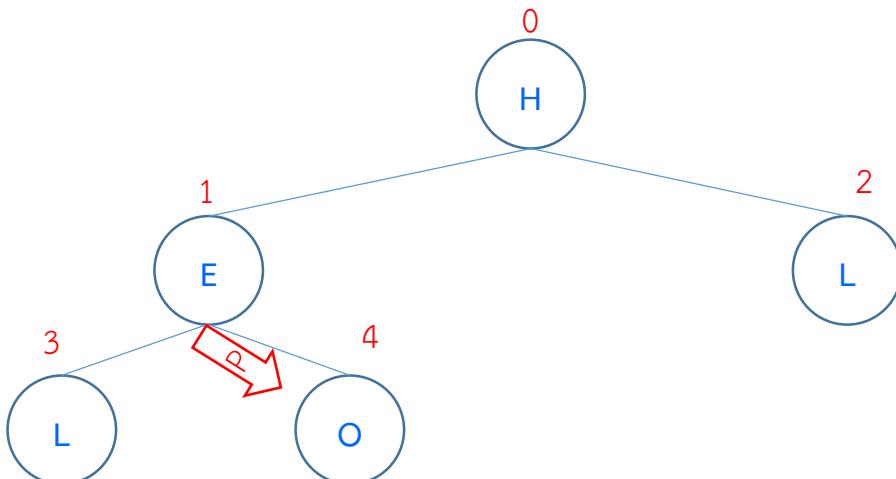


p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L O

Trees: Preorder Traverse Algorithm

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

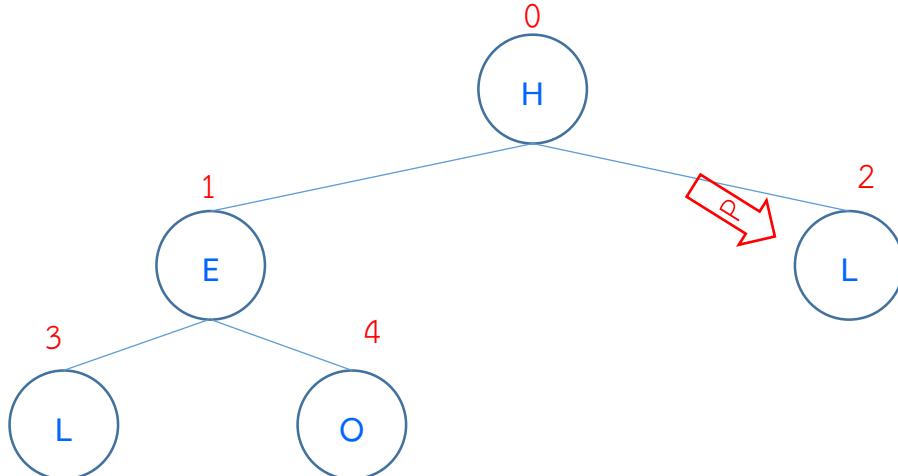


p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L O
2	ว่าง	H E L O

Trees: Preorder Traverse Algorithm

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

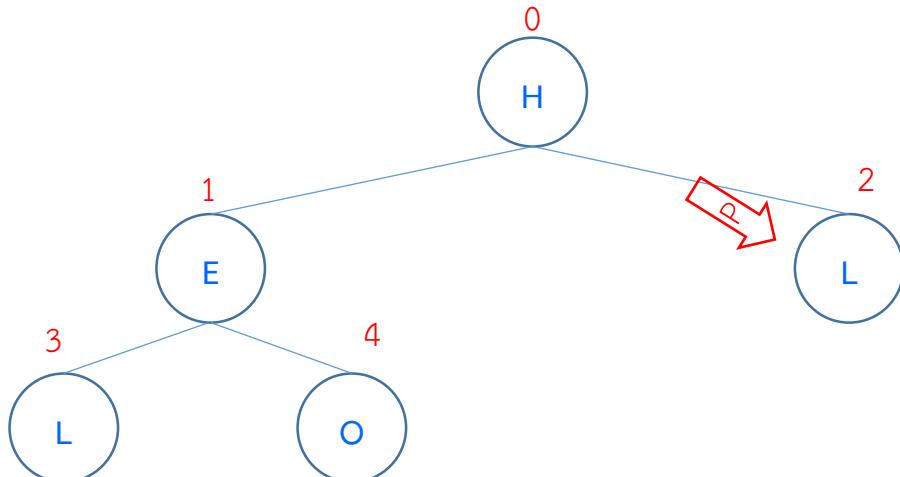


p	Stack	Node ที่ผ่าน
	0	
0	ว่าง	H
0	2	H
1	2	H E
1	4 2	H E
3	4 2	H E L
4	2	H E L O
2	ว่าง	H E L O

Trees: Preorder Traverse Algorithm

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

Trees: Traverse Algorithm

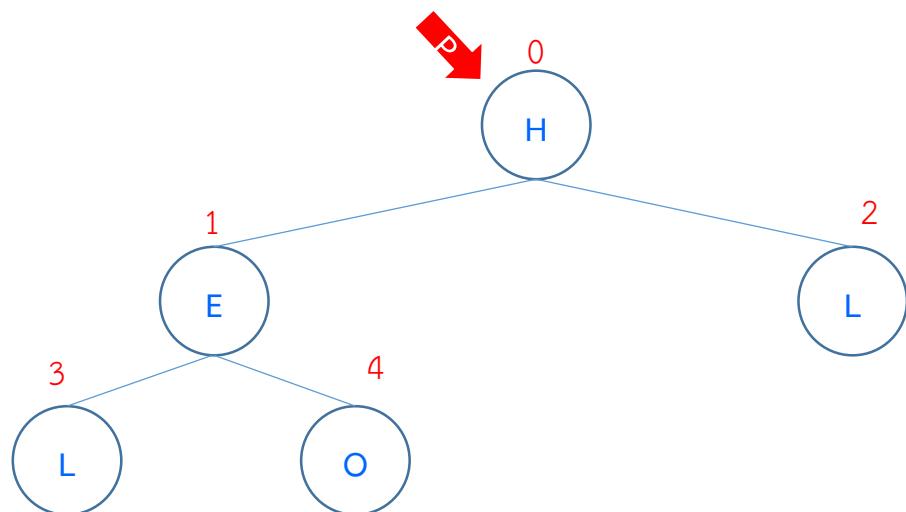
การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

- 1) กำหนด p คือตัวแทนของ root node
- 2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
- 3) ลงมายังตัวแทน subtree ด้านซ้ายของ p เรียกโดย push ตัวแทนของ node ระหว่างทาง จนกว่าจะถึง leaf node
- 4) pop ตัวแทน node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตัวแทนนั้นใน p และวนกลับไปทำข้อ 2

Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

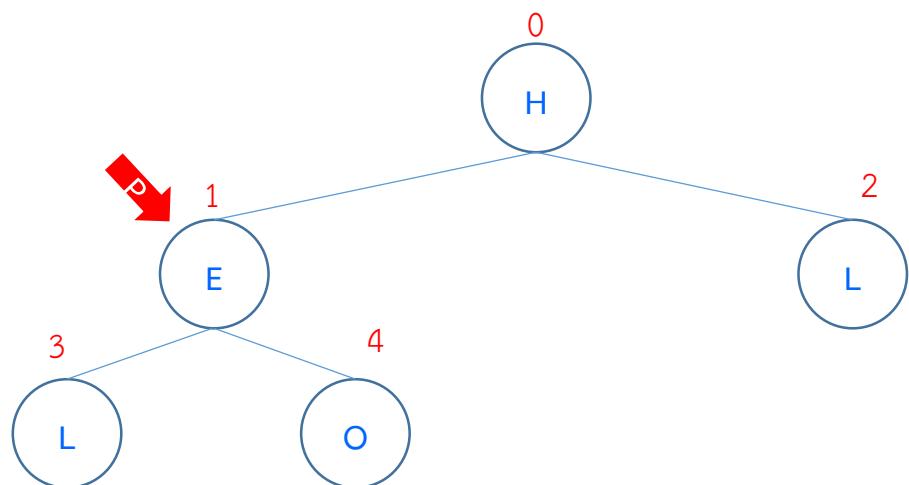
- กำหนด p คือตำแหน่งของ root node
 - ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
 - ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
 - pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมามี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน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



Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

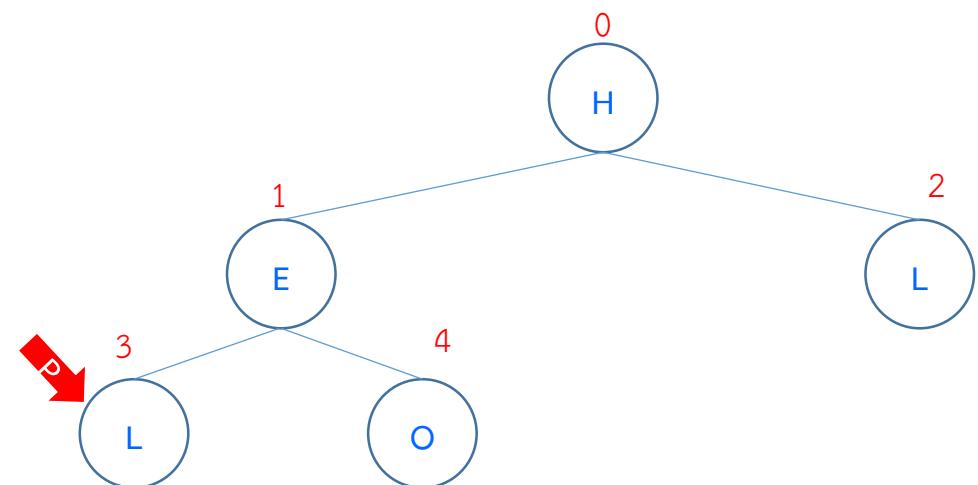
- 1) กำหนด p คือตำแหน่งของ root node
 - 2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
 - 3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
 - 4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมา มี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน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



Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

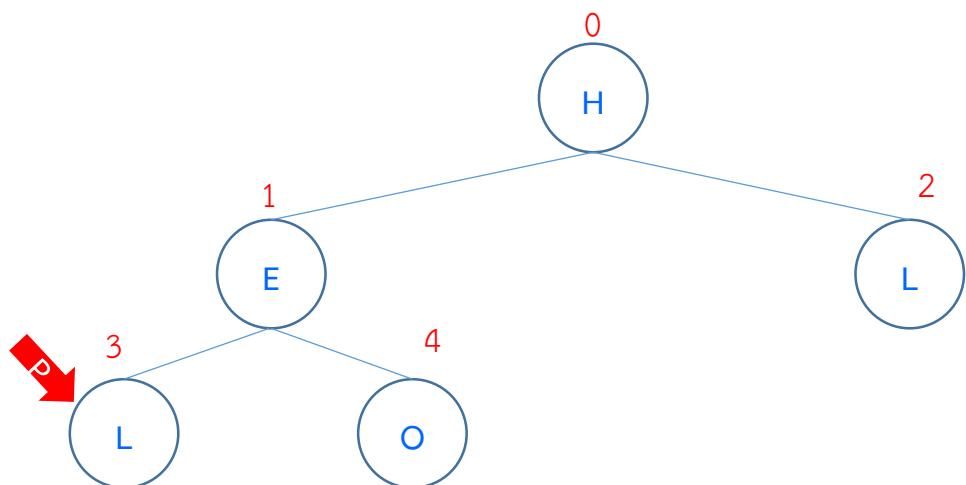
- 1) กำหนด p คือตำแหน่งของ root node
 - 2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
 - 3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
 - 4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมา มี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน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



Trees: Traverse Algorithm

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 ออกมา มี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2

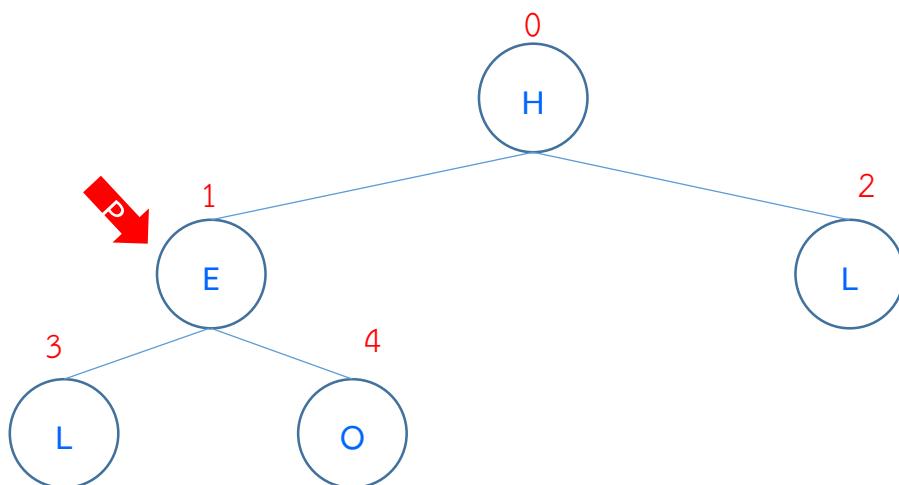


p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L

Trees: Traverse Algorithm

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



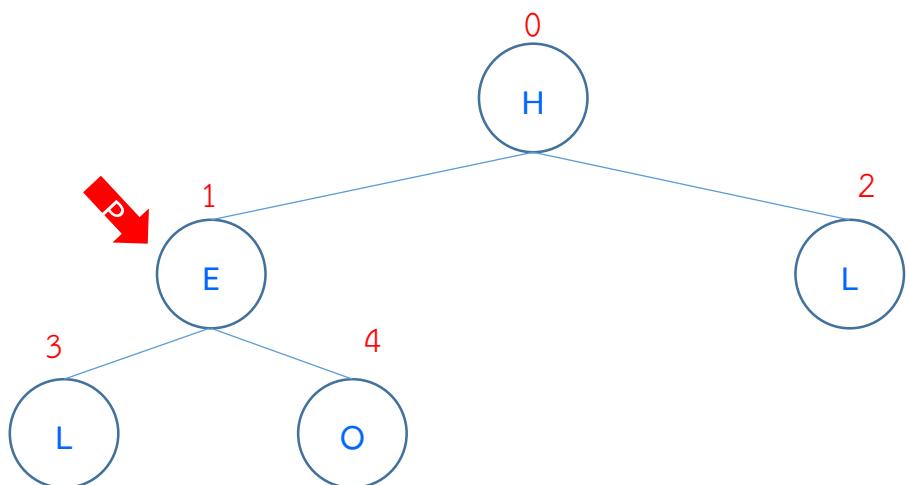
p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E

Trees: Preorder Traverse Algorithm

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

- 1) กำหนด p คือตัวແໜ່ງຂອງ root node
 - 2) ทำຫຼັງນີ້ 3 ແລະ 4 ຈະກະທັງ p ເປັນ null
 - 3) ລົມຍັງຕຳແໜ່ງ subtree ດ້ວຍຫຼາຍຂອງ p ເຮືອຍໂດຍ push
ຕຳແໜ່ງຂອງ node ຮະຫວ່າງທາງ ຈຳກວ່າຈະຄື່ງ leaf node
 - 4) pop ຕຳແໜ່ງ node ອອກມາ ທັງໃຫ້ຈົບໂປຣແກຣມ ທັງ
node ທີ່ pop ອອກມາມີ ກິ່ງລູກທາງດ້ານຂວາ ໃຫ້ໃສຕຳແໜ່ງເນັ້ນໃນ p
ແລະວັນກັບໄປທໍາຂ້ອງ 2

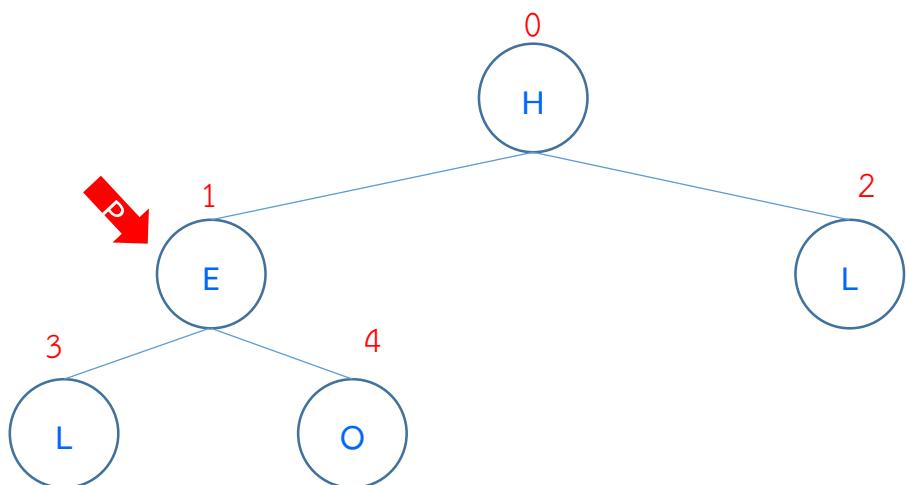
p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E



Trees: Preorder Traverse Algorithm

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

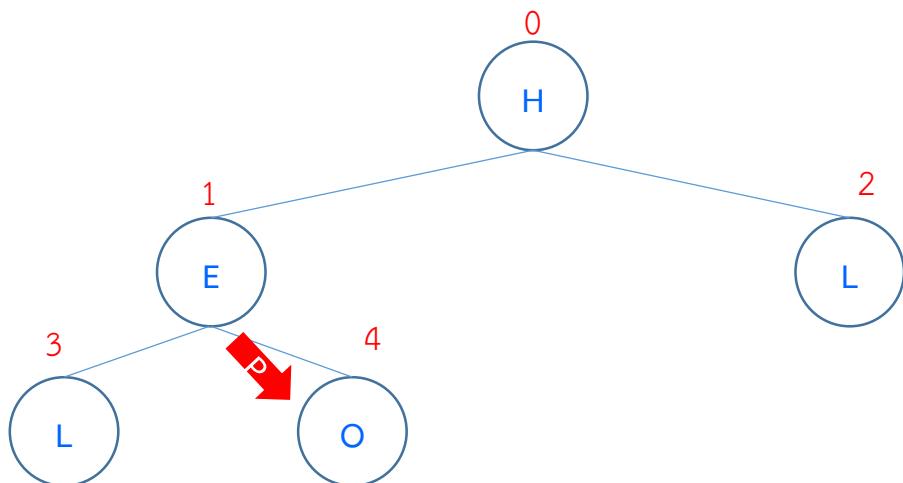
- 1) กำหนด p คือตำแหน่งของ root node
 - 2) ทำซ้ำขั้นตอนที่ 3 และ 4 จนกระทั่ง p เป็น null
 - 3) ลงมายังตำแหน่ง subtree ด้านซ้ายของ p เรื่อยโดย push ตำแหน่งของ node ระหว่างทาง จนกว่าจะถึง leaf node
 - 4) pop ตำแหน่ง node ออกมา หาก stack ว่างให้จบโปรแกรม หาก node ที่ pop ออกมา มี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน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



Trees: Traverse Algorithm

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 ออกมาก็มี กิ่งลูกทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2

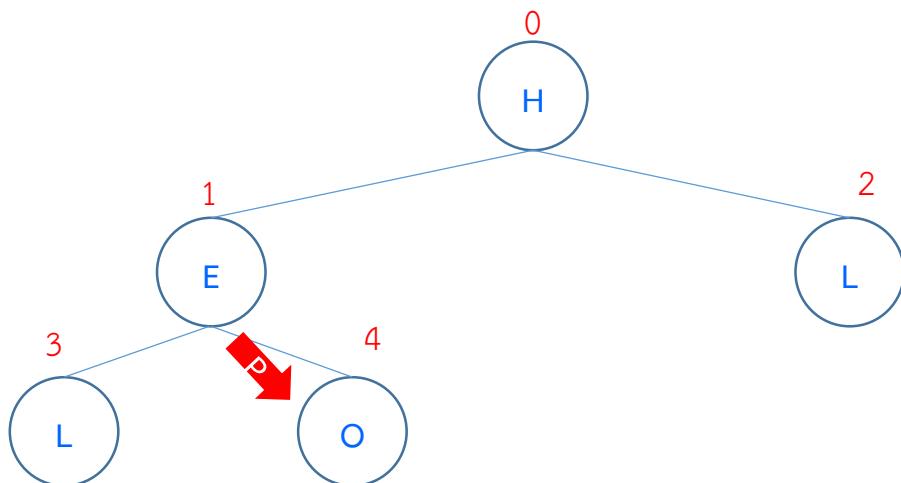


p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E
4	0	L E O

Trees: Traverse Algorithm

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**

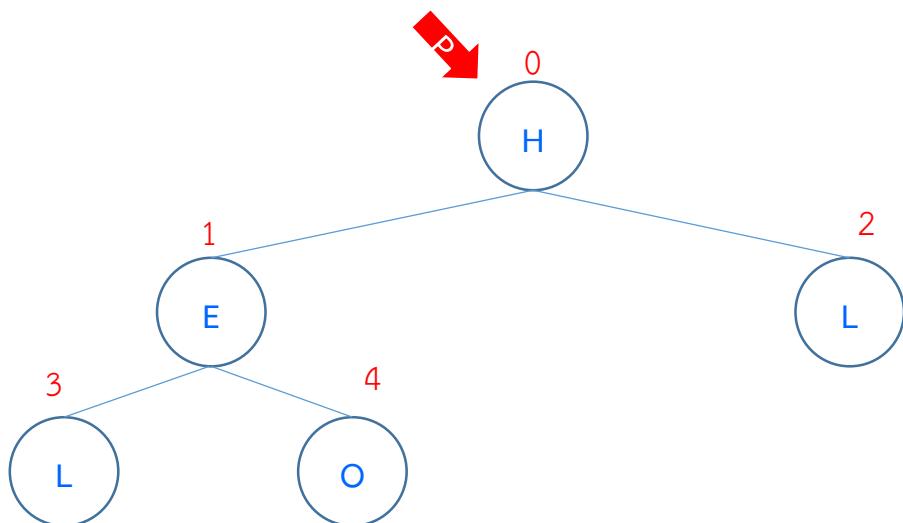


p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E
4	0	L E O
0		L E O

Trees: Traverse Algorithm

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 ออกมาก็ต้องทางด้านขวา ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2

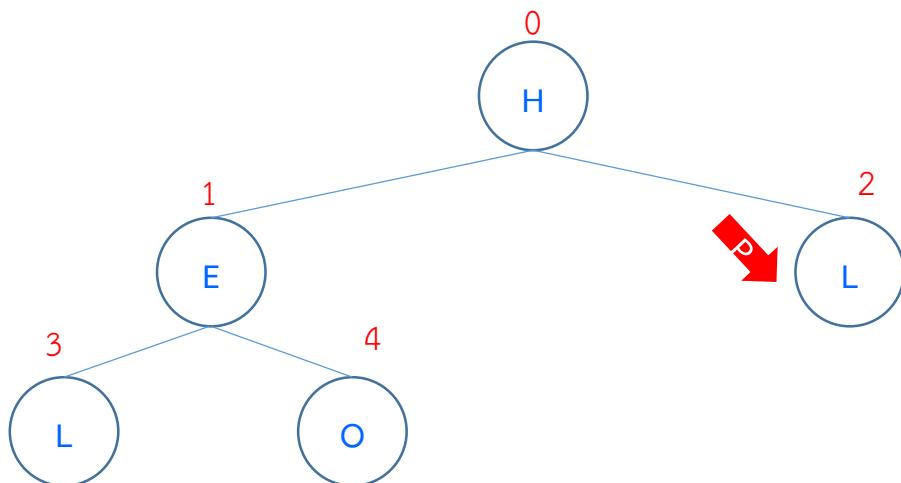


p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E
4	0	L E O
0		L E O
0		L E O H

Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

- 1) กำหนด p คือตำแหน่งของ root node
- 2) **ทำขั้นตอนที่ 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

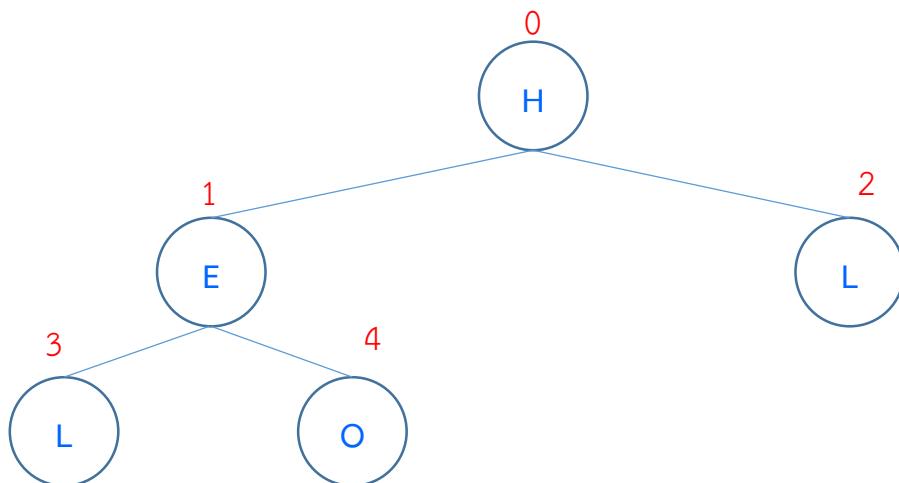


p	Stack	Node ที่ผ่าน
0		
1	0	
3	1 0	L
1	0	L E
1	0	L E
4	0	L E O
0		L E O
0		L E O H
2		L E O H L

Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Inorder แบบใช้ Stack

- 1) กำหนด p คือตำแหน่งของ root node
- 2) ทำขั้นตอนที่ 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

มากกว่า Preorder และ Inorder เพราะขั้นตอนการทำงานของ กิ่งซ้ายและขวาจะไม่เหมือนกัน

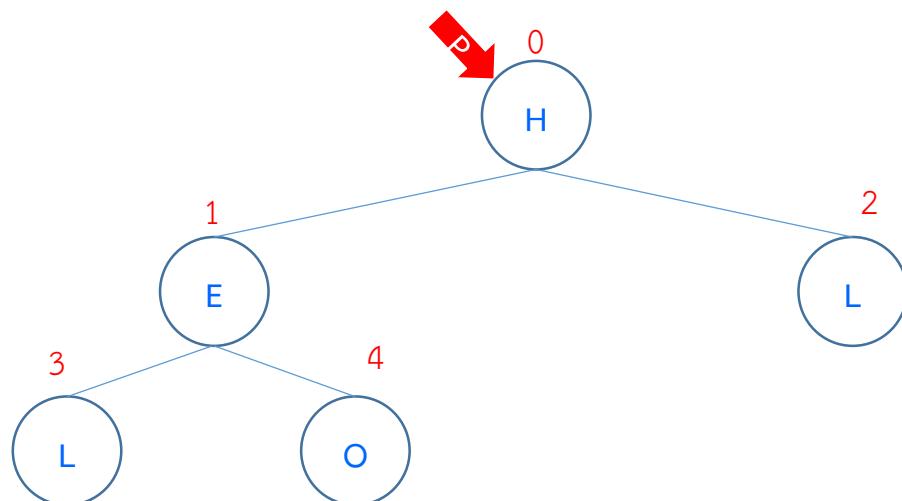
การเก็บ node ข้างจิ้งเพิ่ม \$ ไว้ด้านหน้าตำแหน่ง เพื่อให้รู้ว่าคือ node ขวา

- 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 หาก pb node ที่มี \$ ให้ใส่ตำแหน่งนั้นใน p และวนกลับไปทำข้อ 2

Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

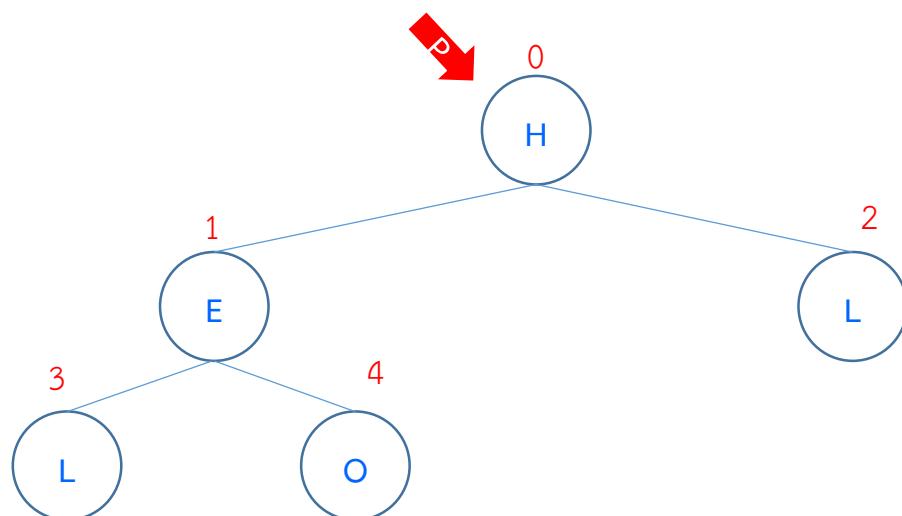
- กำหนด p คือตัวແໜ່ງຂອງ root node และທຳຂັ້ນຕອນ 2 ແລະ 3 ຊຳຈັກວ່າ stack ຈະວ່າງ
 - ໄດ້ລັງມາທາງໜ້າຍເຮືອຍໆຈັກວ່າຈະຄື້ງ leaf node ຮະຫວ່າງທາງໃຫ້ push ຕຳແໜ່ງທີ່ຜ່ານ ພັດມີລູກດ້ານຂວາ ໃຫ້ push ຕຳແໜ່ງລູກໂດຍໄລ່ \$ ໄກສ້າງດ້ານໜ້າ
 - pop ຕຳແໜ່ງທີ່ແມ່ນ \$ ອອກມາທັງໝົດ ພັດ stack ວ່າງ ໃຫ້ຈົບໂປຣແກຣມ ພັດ node ທີ່ມີ \$ ໃຫ້ໄສຕຳແໜ່ງນັ້ນໃນ p ແລະ ວິນກລັບໄປທຳຂ້ອງ 2



Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

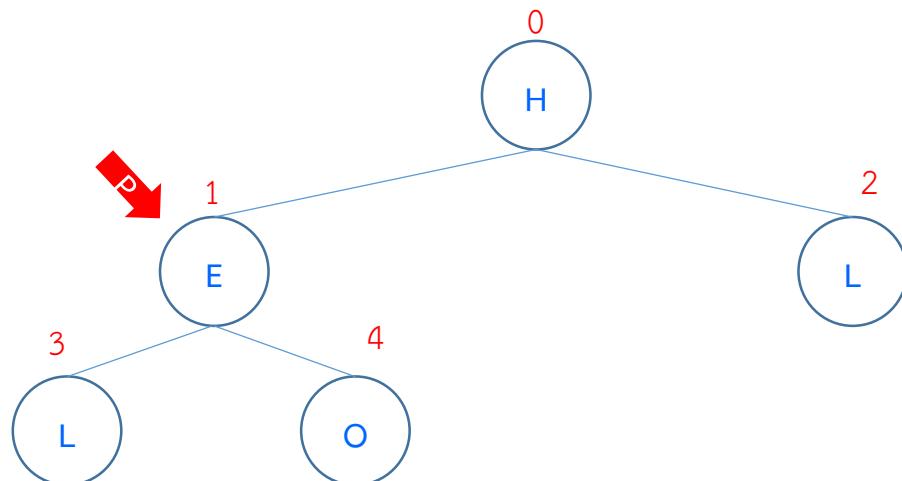
- 1) กำหนด p คือตัวแทนของ root node และทำขั้นตอน 2 และ 3 ซ้ำๆ จนกว่า stack จะว่าง
 - 2) ให้ลงมาทางซ้ายเรื่อยๆ จนกว่าจะถึง leaf node ระหว่างทางให้ push ตัวแทนที่ผ่าน หากมีลูกด้านขวา ให้ push ตัวแทนลูกโดยเลื่อนไปด้านหน้า
 - 3) pop ตัวแทนที่ไม่มี \$ ออกมาทิ้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี \$ ให้ใส่ตัวแทนนั้นใน p และวนกลับไปทำข้อ 2



Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

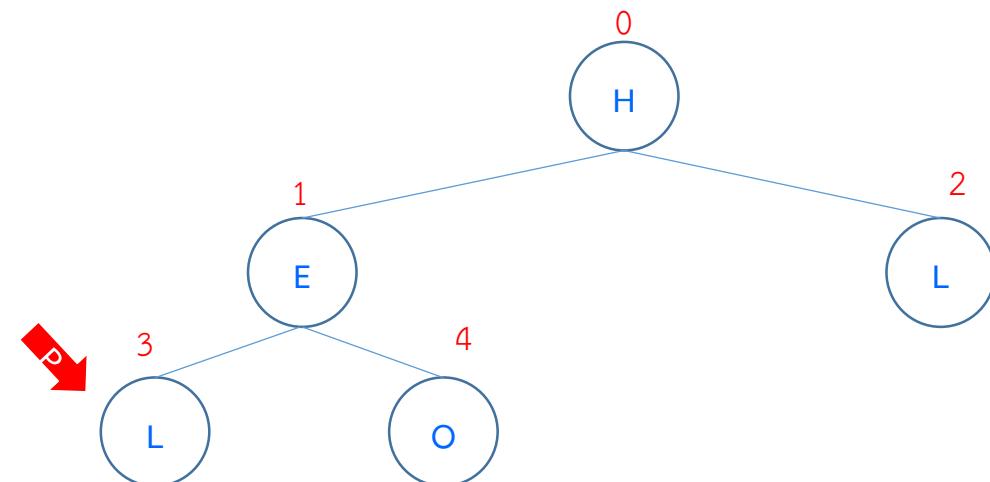
- 1) กำหนด p คือตัวแทนของ root node และทำขั้นตอน 2 และ 3 ซ้ำๆ จนกว่า stack จะว่าง
 - 2) ให้ลงมาทางซ้ายเรื่อยๆ จนกว่าจะถึง leaf node ระหว่างทางให้ push ตัวแทนที่ผ่าน หากมีลูกด้านขวา ให้ push ตัวแทนลูกโดยเลื่อนไปด้านหน้า
 - 3) pop ตัวแทนที่ไม่มี \$ ออกมาทิ้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี \$ ให้ใส่ตัวแทนนั้นใน p และวนกลับไปทำข้อ 2



Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

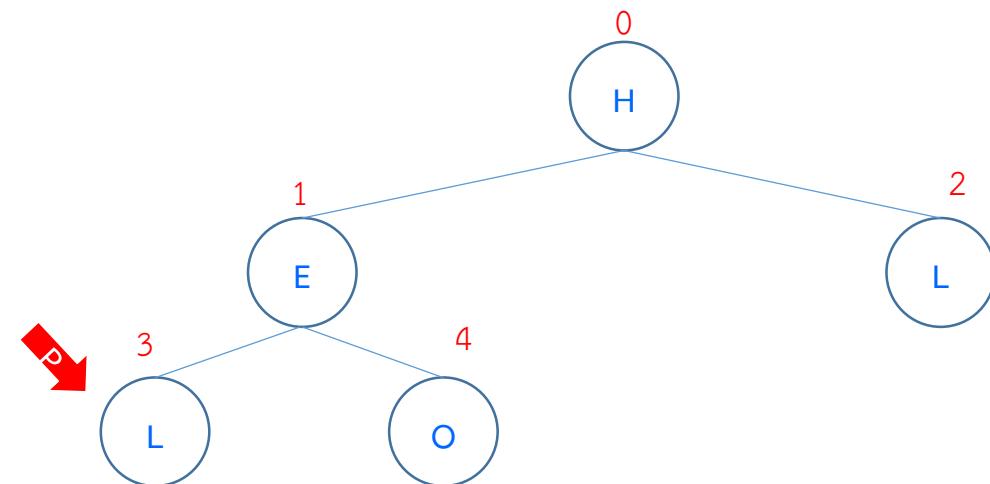
- 1) กำหนด p คือตัวแทนของ root node และทำขั้นตอน 2 และ 3 ซ้ำๆ จนกว่า stack จะว่าง
 - 2) ให้ลงมาทางซ้ายเรื่อยๆ จนกว่าจะถึง leaf node ระหว่างทางให้ push ตัวแทนที่ผ่าน หากมีลูกด้านขวา ให้ push ตัวแทนลูกโดยเลื่อนไปด้านหน้า
 - 3) pop ตัวแทนที่ไม่มี \$ ออกมาทิ้งหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี \$ ให้ใส่ตัวแทนนั้นใน p และวนกลับไปทำข้อ 2



Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

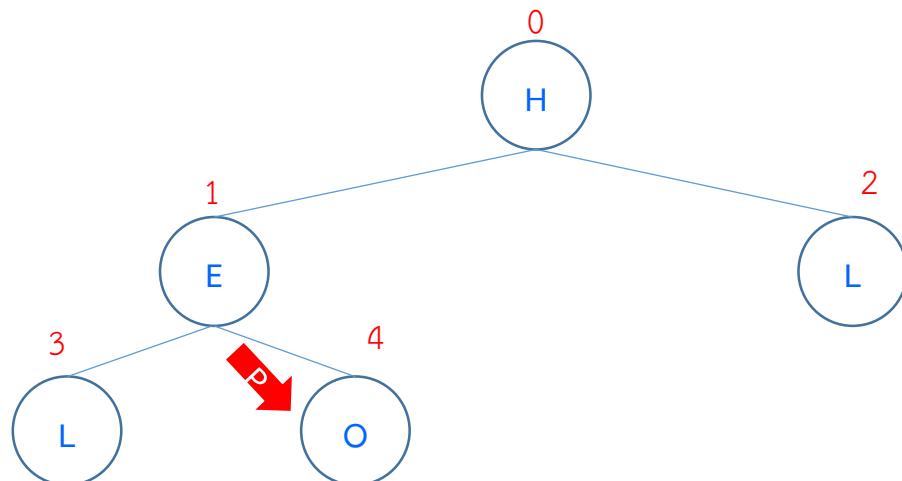
- 1) กำหนด p คือตัวแทนของ root node และทำขั้นตอน 2 และ 3 ซ้ำๆ จนกว่า stack จะว่าง
 - 2) ใส่ลงมาทางซ้ายเรื่อยๆ จนกว่าจะถึง leaf node ระหว่างทางให้ push ตัวแทนที่ผ่าน หากมีลูกด้านขวา ให้ push ตัวแทนลูกโดยเลื่อนไปด้านหน้า
 - 3) pop ตัวแทนที่ไม่มี \$ ออกมาก้างหนด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี \$ ให้ใส่ตัวแทนนั้นใน p และวนกลับไปทำข้อ 2



Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

- 1) กำหนด p คือตัวແໜ່ງຂອງ root node และທຳບັນດາອຸປະກອນ 2 ແລະ 3 ຊໍາຈົກວ່າ stack ຈະວ່າງ
 - 2) ໄຕ່ລັງມາທາງໜ້າຍເຮືອຍໆຈົກວ່າຈະຄື່ງ leaf node ຮະຫວ່າງທາງໃໝ່ push ຕຳແໜ່ງທີ່ຜ່ານ ຫາກມີລູກດ້ານຂວາ ໃຫ້ push ຕຳແໜ່ງລູກໂດຍໃສ່ \$ ໄວດ້ານໜ້າ
 - 3) pop ຕຳແໜ່ງທີ່ໄມ່ມີ \$ ອອກມາທັງໝົດ ຫາກ stack ວ່າງ ໃຫ້ຈົບໂປຣແກຣມ ຫາກພບ node ທີ່ມີ \$ ໃຫ້ໃສ່ຕຳແໜ່ງນັ້ນໃນ p ແລະ ວນກລັບໄປທຳຂ້ອງ 2

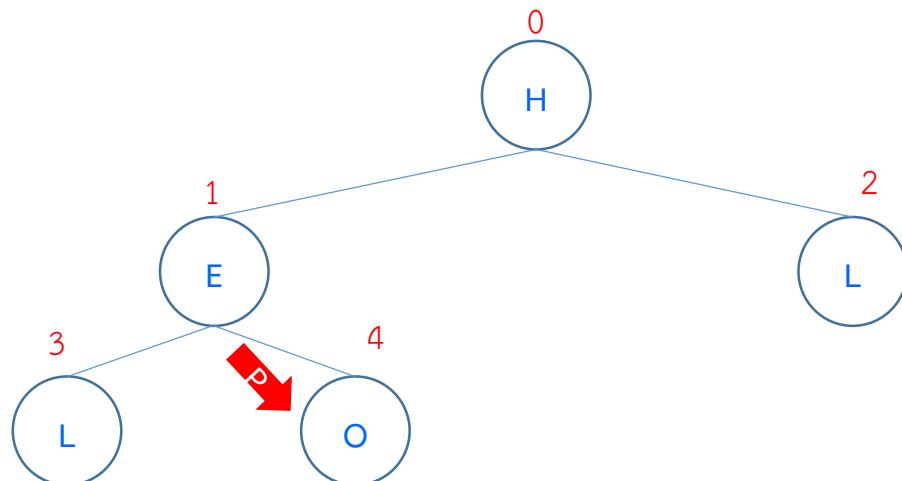


p	Stack	Node ที่ผ่าน
0		
0	\$2 0	
1	\$2 0	
1	\$4 1 \$2 0	
3	\$4 1 \$2 0	L
\$4	1 \$2 0	

Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

- 1) กำหนด p คือตัวແໜ່ງຂອງ root node และทำຫຸ້ນຕອນ 2 ແລະ 3 ຊໍາຈັກວ່າ stack ຈະວ່າງ
 - 2) ໄຕ່ລັງມາທາງໜ້າຍເຮືອຍໆຈັກວ່າຈະຄື້ງ leaf node ຮະຫວ່າງທາງໃໝ່ push ຕຳແໜ່ງທີ່ຜ່ານ ຫາກມີລູກດ້ານຂວາ ໃຫ້ push ຕຳແໜ່ງລູກໂດຍໃສ່ \$ ໄວດ້ານໜ້າ
 - 3) pop ຕຳແໜ່ງທີ່ໄມ່ມີ \$ ອອກມາທີ່ໜັດ ຫາກ stack ວ່າງ ໃຫ້ຈົບໂປຣແກຣມ ຫາກພບ node ທີ່ມີ \$ ໃຫ້ໃສ່ຕຳແໜ່ງນັ້ນໃນ p ແລະ ວນກລັບໄປທຳຂ້ອງ 2

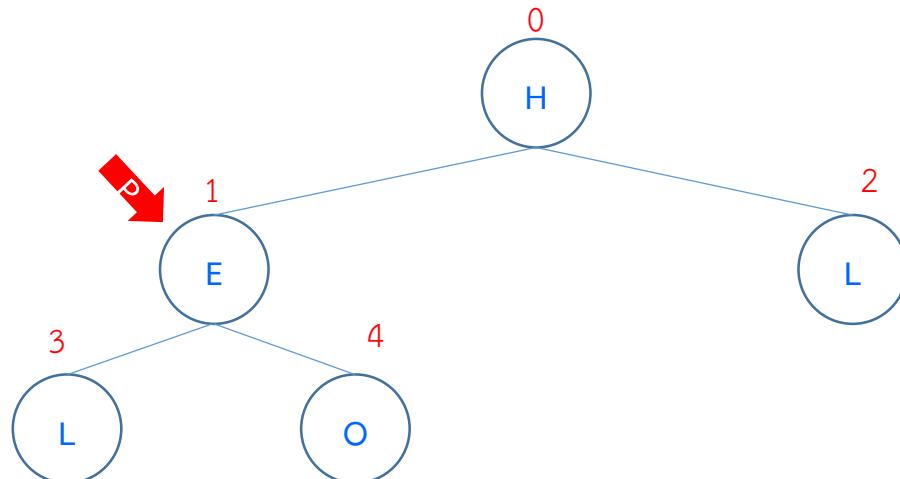


p	Stack	Node ที่ผ่าน
0		
0	\$2 0	
1	\$2 0	
1	\$4 1 \$2 0	
3	\$4 1 \$2 0	L
\$4	1 \$2 0	L O

Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

- 1) กำหนด p คือตัวแทนของ root node และทำขั้นตอน 2 และ 3 ซ้ำๆ จนกว่า stack จะว่าง
 - 2) ไต่ลงมาทางซ้ายเรื่อยๆ จนกว่าจะถึง leaf node ระหว่างทางให้ push ตัวแทนที่ผ่าน หากมีลูกด้านขวา ให้ push ตัวแทนลูกโดยใส่ \$ ไว้ด้านหน้า
 - 3) pop ตัวแทนที่ไม่มี \$ ออกมาก้างหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี \$ ให้ใส่ตัวแทนนั้นใน p และวนกลับไปทำข้อ 2

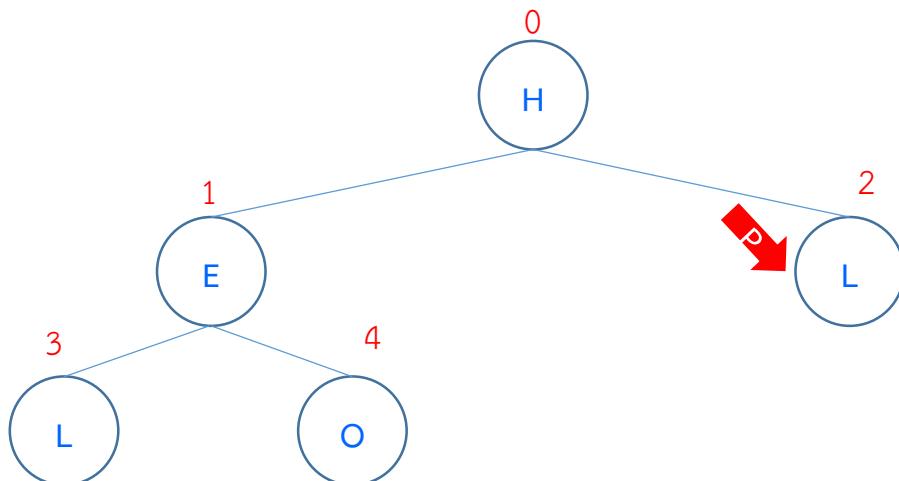


p	Stack	Node ที่ผ่าน
0		
0	\$2 0	
1	\$2 0	
1	\$4 1 \$2 0	
3	\$4 1 \$2 0	L
\$4	1 \$2 0	L O
1	0	L O E

Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

- 1) กำหนด p คือตัวແໜ່ງຂອງ root node และທຳບັນດາອຸປະກອນ 2 ແລະ 3 ຊຳຈັກວ່າ stack ຈະວ່າງ
 - 2) ໄຕ່ລັງມາທາງໜ້າຍເຮືອຍໆຈັກວ່າຈະຄື່ງ leaf node ຮະຫວ່າງທາງໃໝ່ push ຕຳແໜ່ງທີ່ຜ່ານ ຫາກມີລູກດ້ານຂວາ ໃຫ້ push ຕຳແໜ່ງລູກໂດຍໃສ່ \$ ໄວດ້ານໜ້າ
 - 3) pop ຕຳແໜ່ງທີ່ໄມ່ມີ \$ ອອກມາທັງໝົດ ຫາກ stack ວ່າງ ໃຫ້ຈົບໂປຣແກຣມ ຫາກພບ node ທີ່ມີ \$ ໃຫ້ໃສ່ຕຳແໜ່ງນັ້ນໃນ p ແລະ ວນກລັບໄປທຳຂ້ອງ 2

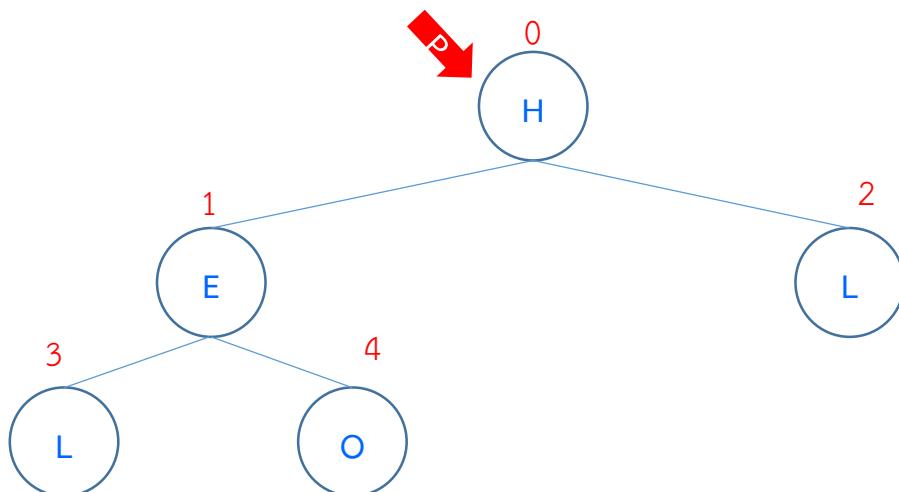


p	Stack	Node ที่ผ่าน
0		
0	\$2 0	
1	\$2 0	
1	\$4 1 \$2 0	
3	\$4 1 \$2 0	L
\$4	1 \$2 0	L O
1	1 0	L O E
\$2	0	L O E L

Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

- 1) กำหนด p คือตำแหน่งของ root node และทำขั้นตอน 2 และ 3 ซ้ำ
จนกว่า stack จะว่าง
- 2) ให้ลงมาทางซ้ายเรื่อยๆจนกว่าจะถึง leaf node ระหว่างทางให้
push ตำแหน่งที่ผ่าน หากมีลูกด้านขวา ให้ push ตำแหน่งลูกโดยใส่
\$ ไว้ด้านหน้า
- 3) pop ตำแหน่งที่ไม่มี \$ ออกมาก้างหมด หาก stack ว่าง ให้จบ
โปรแกรม หากพบร node ที่มี \$ ให้ใส่ตำแหน่งนั้นใน p และวน
กลับไปทำข้อ 2

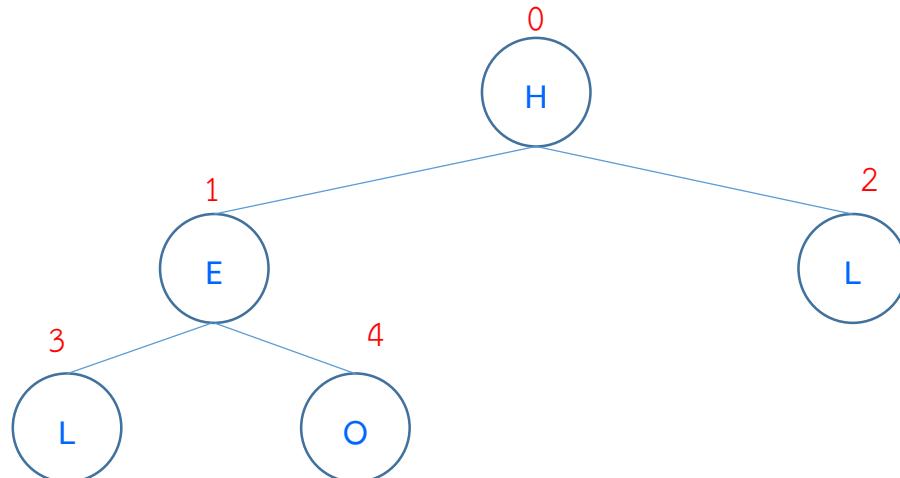


p	Stack	Node ที่ผ่าน
0		
0	\$2 0	
1	\$2 0	
1	\$4 1 \$2 0	
3	\$4 1 \$2 0	L
\$4	1 \$2 0	L O
1	0	L O E
\$2	0	L O E L
0		L O E L

Trees: Traverse Algorithm

การท่องไปในต้นไม้แบบ Postorder แบบใช้ Stack

- 1) กำหนด p คือตัวแทนของ root node และทำขั้นตอน 2 และ 3 ซ้ำๆ จนกว่า stack จะว่าง
 - 2) ไต่ลงมาทางซ้ายเรื่อยๆ จนกว่าจะถึง leaf node ระหว่างทางให้ push ตัวแทนที่ผ่าน หากมีลูกด้านขวา ให้ push ตัวแทนลูกโดยใส่ \$ ไว้ด้านหน้า
 - 3) pop ตัวแทนที่ไม่มี \$ ออกมาก้างหมด หาก stack ว่าง ให้จบโปรแกรม หากพบ node ที่มี \$ ให้ใส่ตัวแทนนั้นใน 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

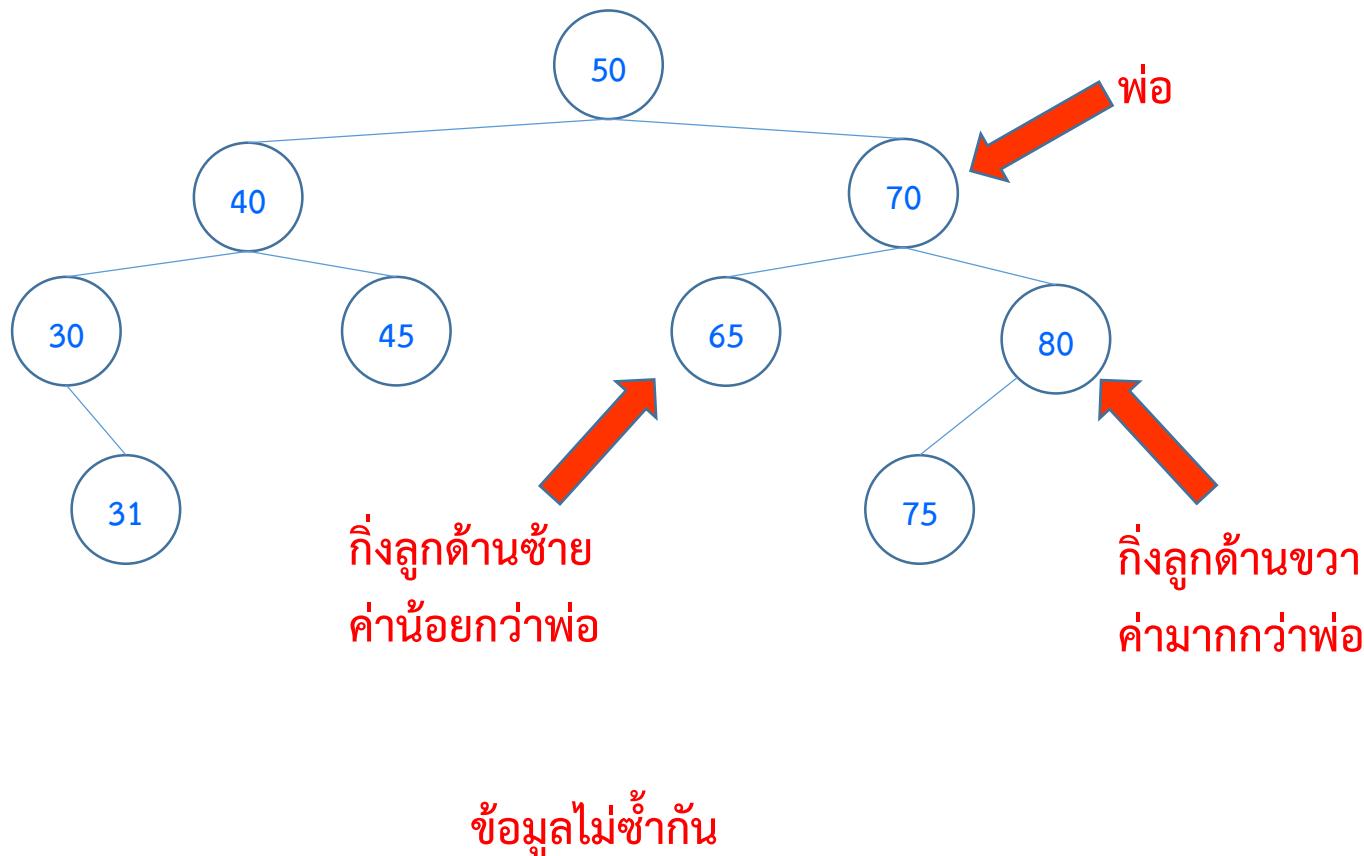
การสร้าง เพิ่ม หรือลบ ข้อมูลออกจากต้นไม้
จำเป็นต้อง รู้วิธีการ Traverse ก่อน จึงจะสามารถ ทำ operation เหล่านี้ได้

ย้อนกลับไปยังต้นไม่ทวิภาค ต้นแรกในบทนี้

Binary Search Tree

Binary Search Trees

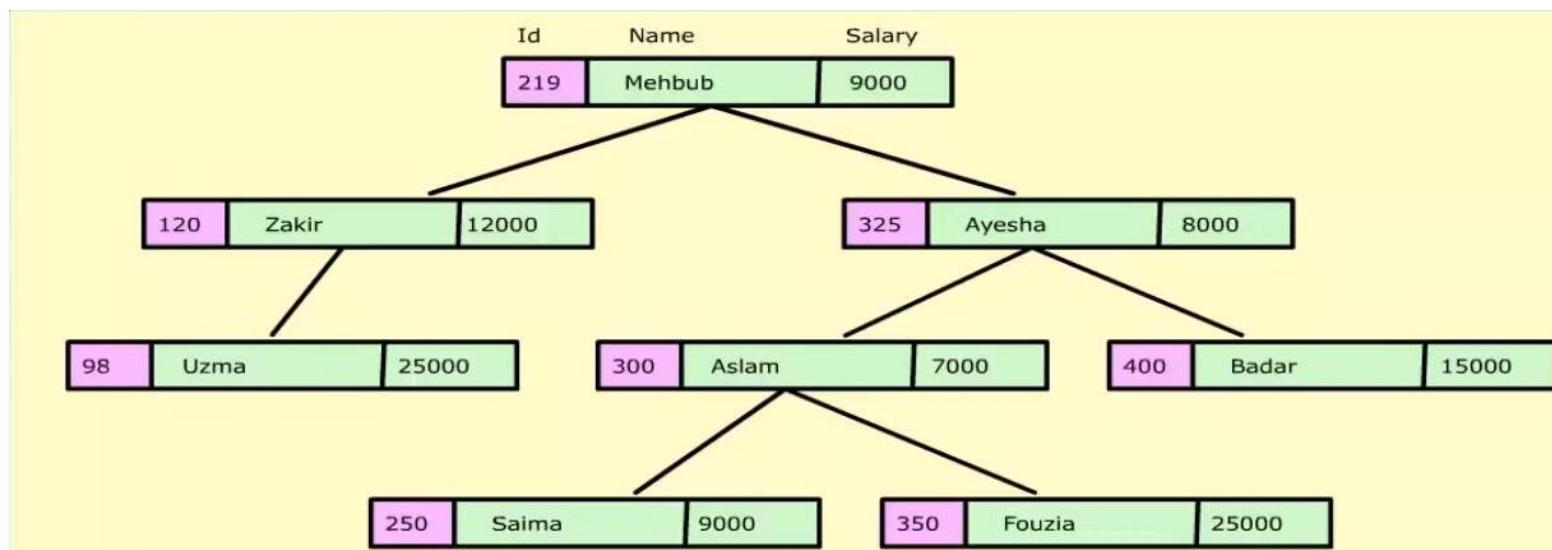
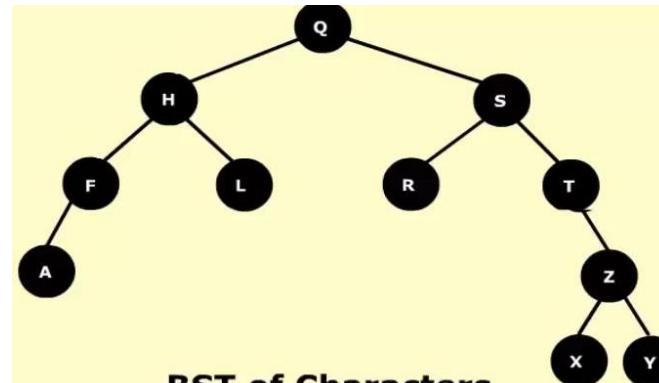
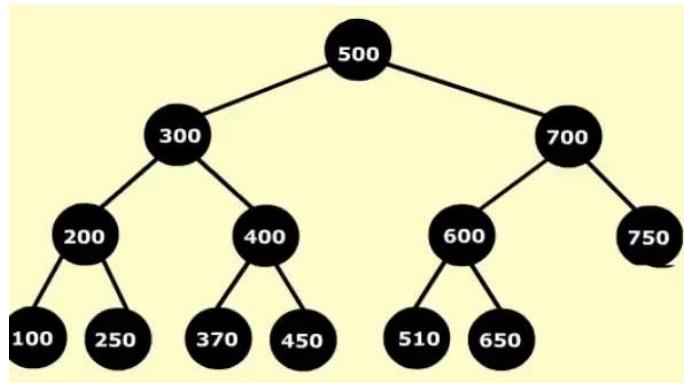
- ค่าของ node จะซ้ำกันไม่ได้
- กิ่งลูกทางด้านซ้ายจะต้องมีค่าน้อยกว่าพ่อ และกิ่งลูกทางด้านขวาต้องมีค่ามากกว่าพ่อ เสมอ
- ข้อมูลในต้นไม้ต้องเป็นแบบที่สามารถเปรียบเทียบกันได้เท่านั้น



Trees: Binary Search Trees

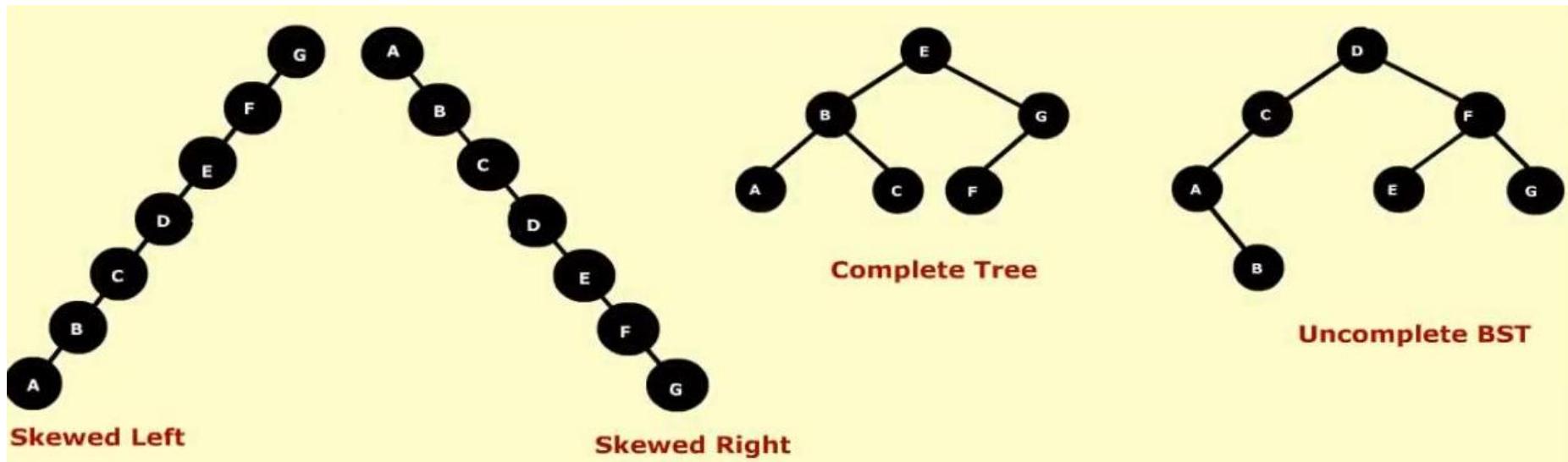
ข้อมูลในแต่ละ node อาจเป็นจำนวน หรือ ตัวอักษรก็ได้

หากต้องการเก็บข้อมูลที่ซับซ้อนกว่านั้น ให้สร้าง Structure และกำหนด ID ของ Node



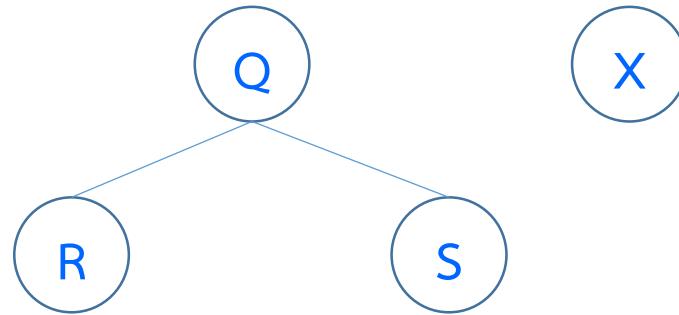
Trees: Binary Search Trees

ข้อมูลเดียวกัน แต่ลำดับไม่เหมือนกัน จะทำให้เกิดต้นไม้ที่ต่างกัน



Trees: Binary Search Trees

การเพิ่มข้อมูลใน Binary Search Tree

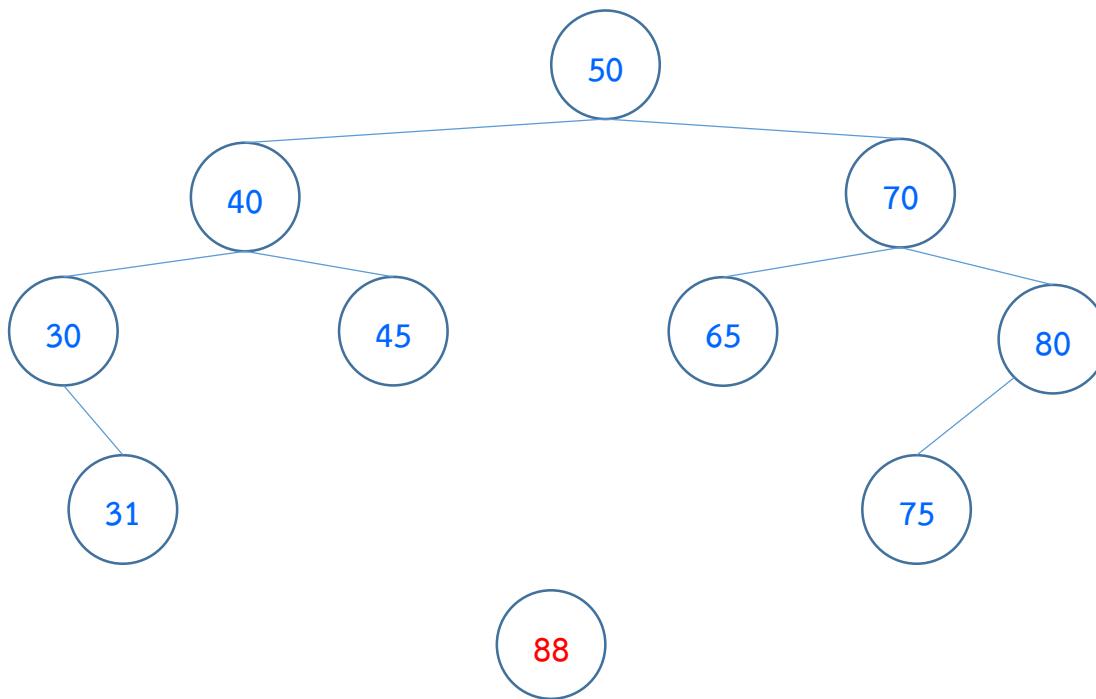


- 1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
- 2) หากค่าน้อยกว่า Root ให้ไปทางซ้าย หากมากกว่าให้ไปทางขวา
- 3) ทำซ้ำไปเรื่อย ๆ จนกว่าจะถึง node สุดท้าย
- 4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา

Trees: Binary Search Trees

การเพิ่มข้อมูลใน Binary Search Tree

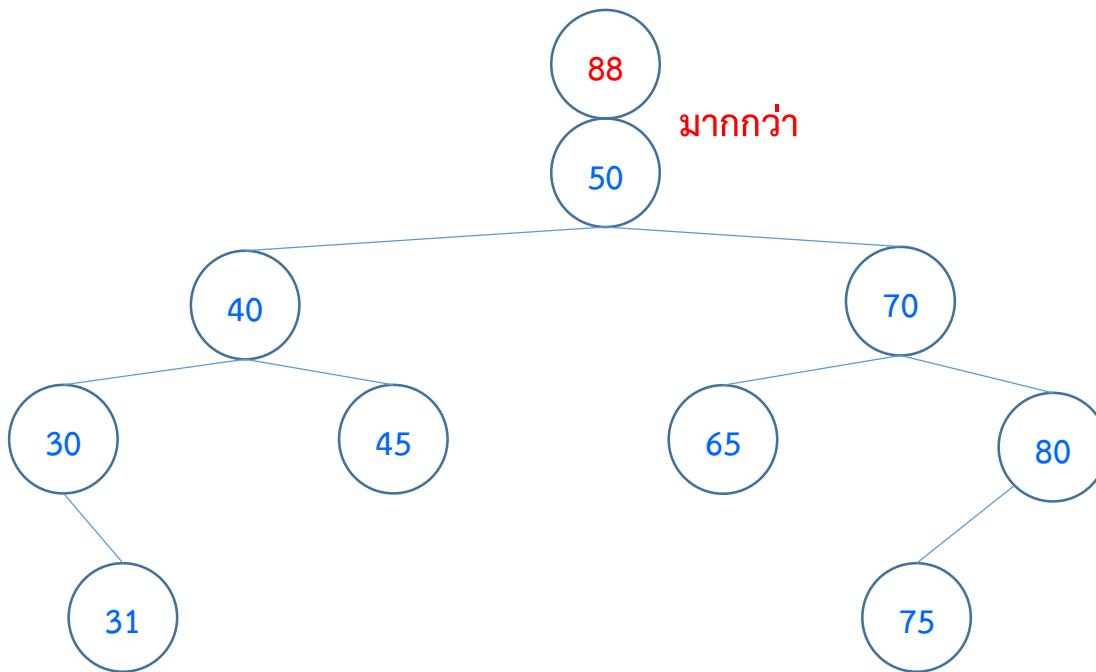
- 1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
- 2) หากค่าน้อยกว่า Root ให้ไปทางซ้าย หากมากกว่าให้ไปทางขวา
- 3) ทำซ้ำไปเรื่อยๆ จนกว่าจะถึง node สุดท้าย
- 4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา



Trees: Binary Search Trees

การเพิ่มข้อมูลใน Binary Search Tree

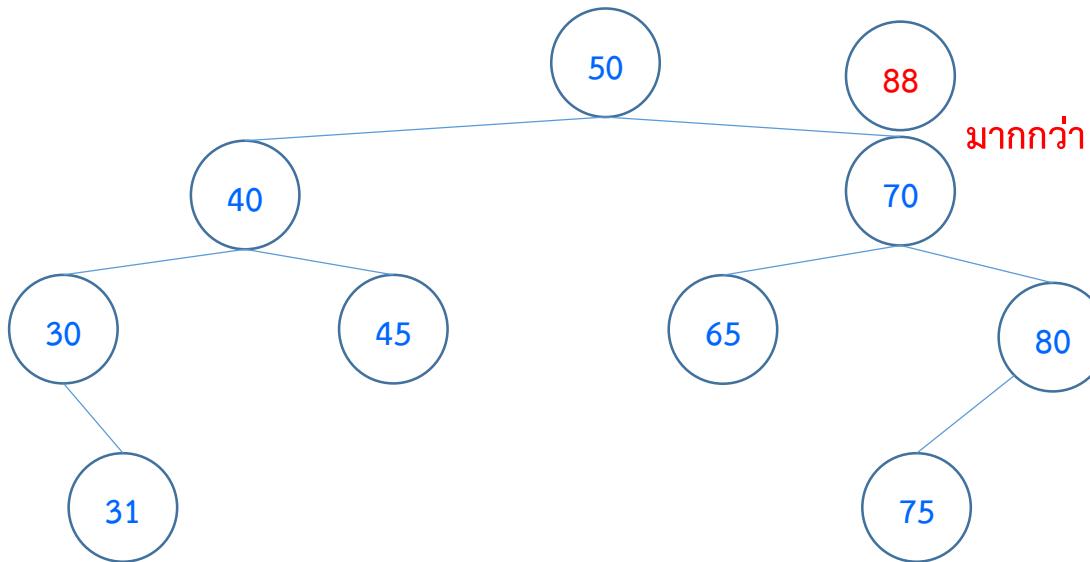
- 1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
- 2) หากค่าน้อยกว่า Root ให้ไปทางซ้าย หากมากกว่าให้ไปทางขวา
- 3) ทำซ้ำไปเรื่อยๆ จนกว่าจะถึง node สุดท้าย
- 4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา



Trees: Binary Search Trees

การเพิ่มข้อมูลใน Binary Search Tree

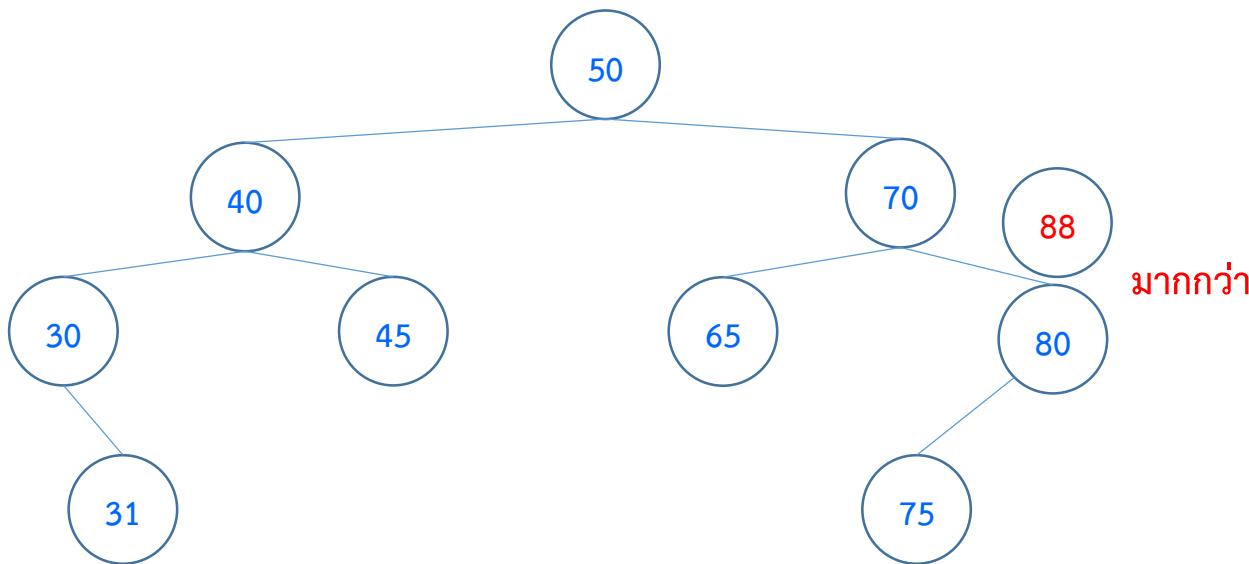
- 1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
- 2) หากค่าน้อยกว่า Root ให้ไปทางซ้าย หากมากกว่าให้ไปทางขวา
- 3) ทำซ้ำไปเรื่อยๆ จนกว่าจะถึง node สุดท้าย
- 4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา



Trees: Binary Search Trees

การเพิ่มข้อมูลใน Binary Search Tree

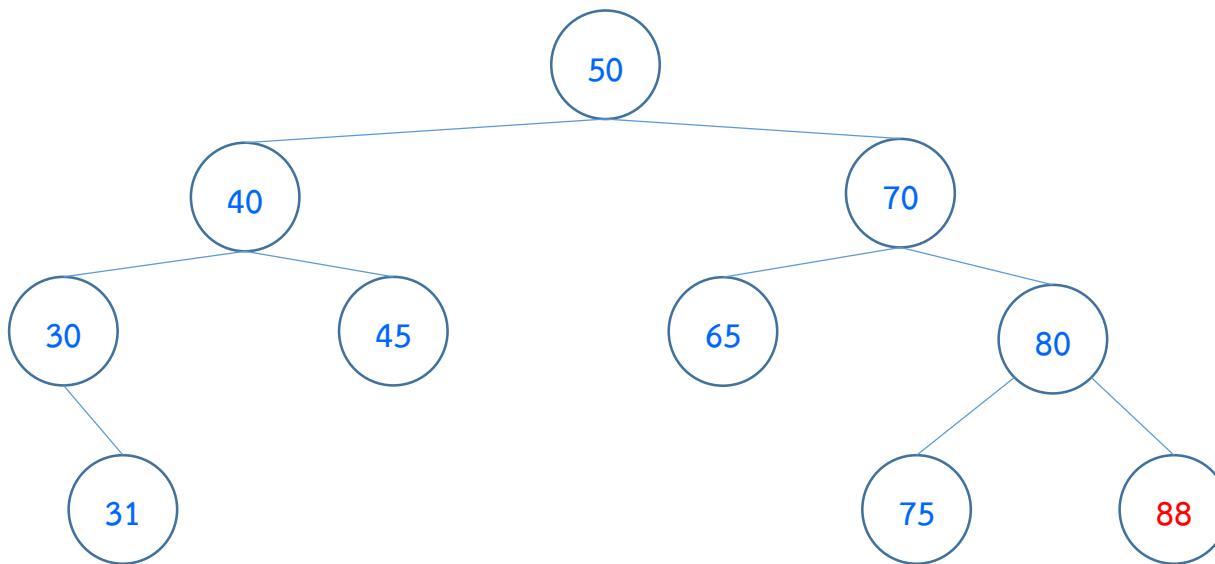
- 1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
- 2) หากค่าน้อยกว่า Root ให้ไปทางซ้าย หากมากกว่าให้ไปทางขวา
- 3) ทำซ้ำไปเรื่อยๆ จนกว่าจะถึง node สุดท้าย
- 4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา



Trees: Binary Search Trees

การเพิ่มข้อมูลใน Binary Search Tree

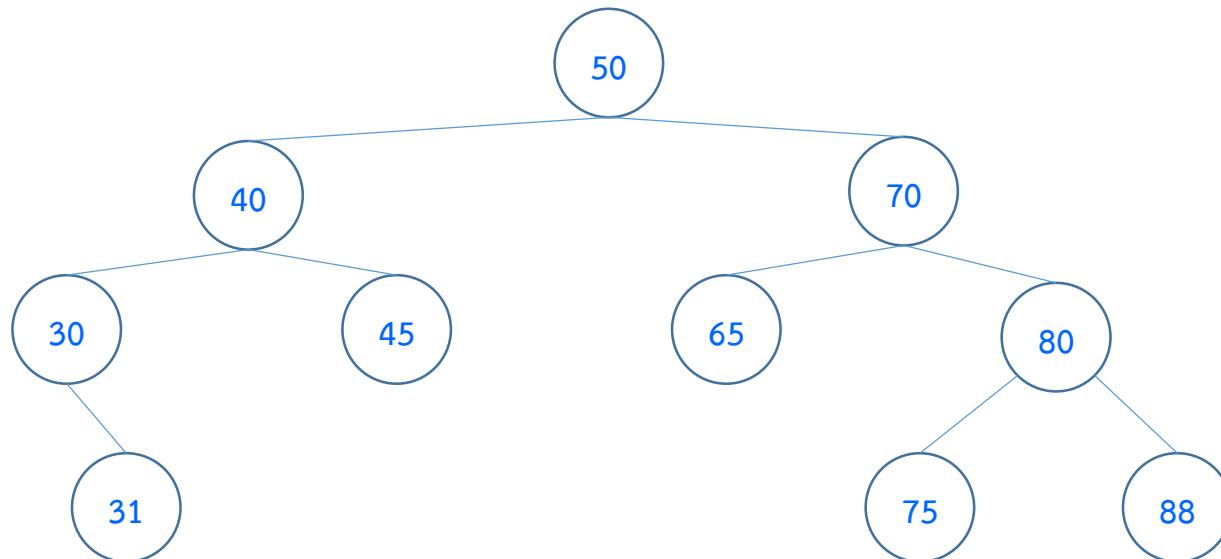
- 1) เริ่มจาก Root node ทำการเปรียบเทียบ node ที่สร้างใหม่กับ Root
- 2) หากค่าน้อยกว่า Root ให้ไปทางซ้าย หากมากกว่าให้ไปทางขวา
- 3) ทำซ้ำไปเรื่อยๆ จนกว่าจะถึง node สุดท้าย
- 4) หากค่าน้อยกว่า node สุดท้ายให้กำหนด node ใหม่เป็นลูกทางด้านซ้าย หากมากกว่าให้เป็นลูกด้านขวา



Trees: Binary Search Trees

การค้นข้อมูลใน Binary Search Tree

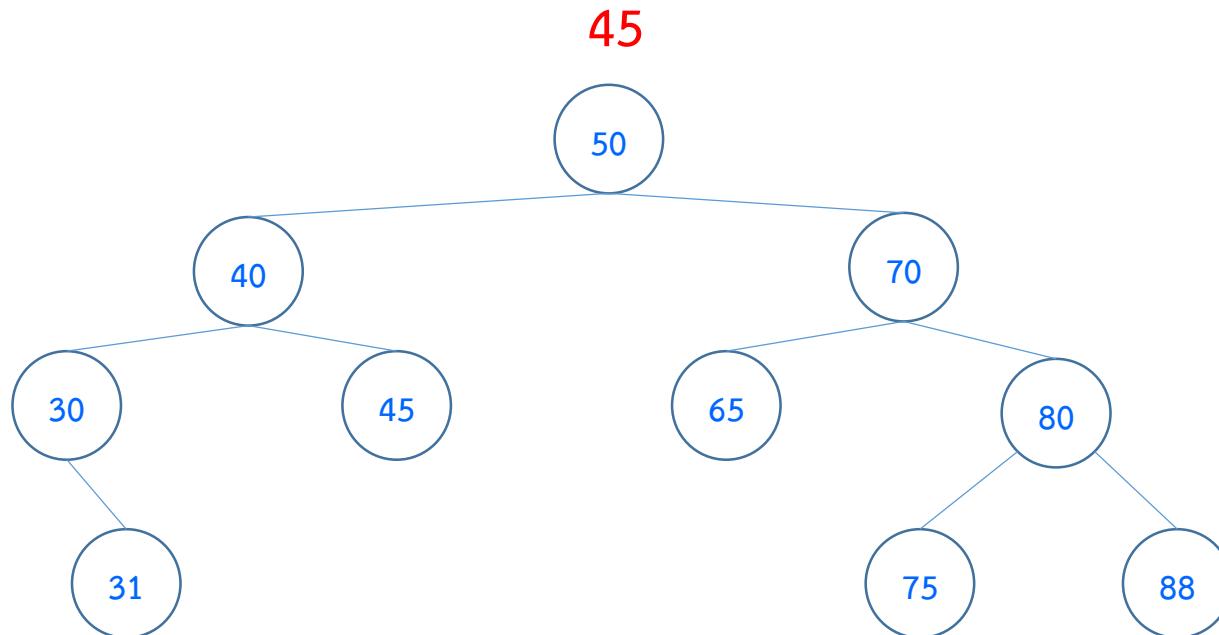
การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



Trees: Binary Search Trees

การค้นข้อมูลใน Binary Search Tree

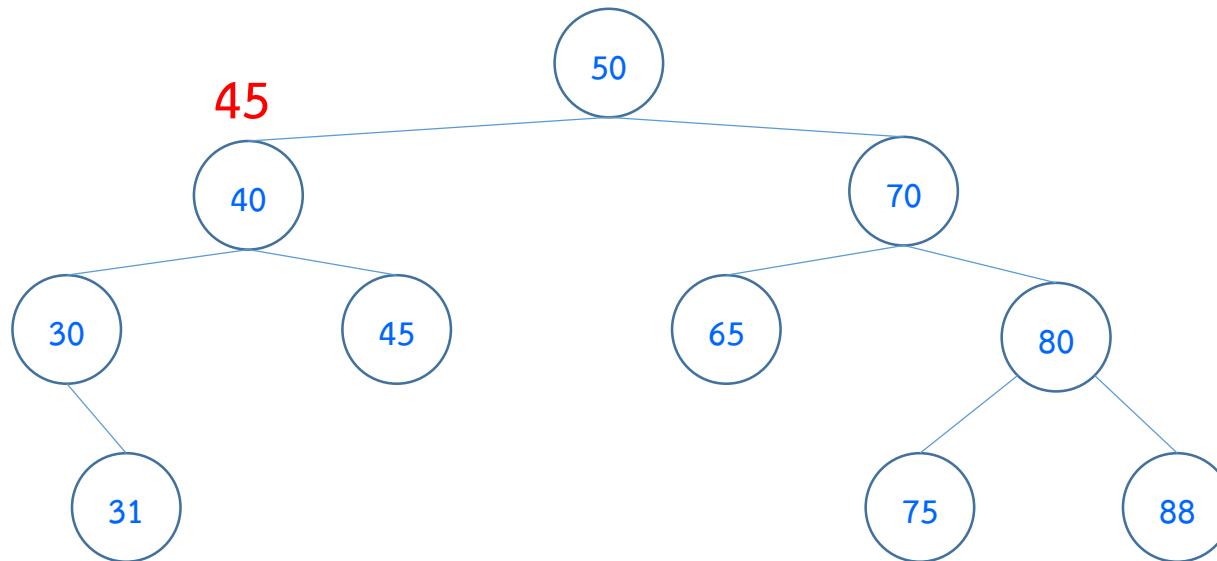
การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



Trees: Binary Search Trees

การค้นข้อมูลใน Binary Search Tree

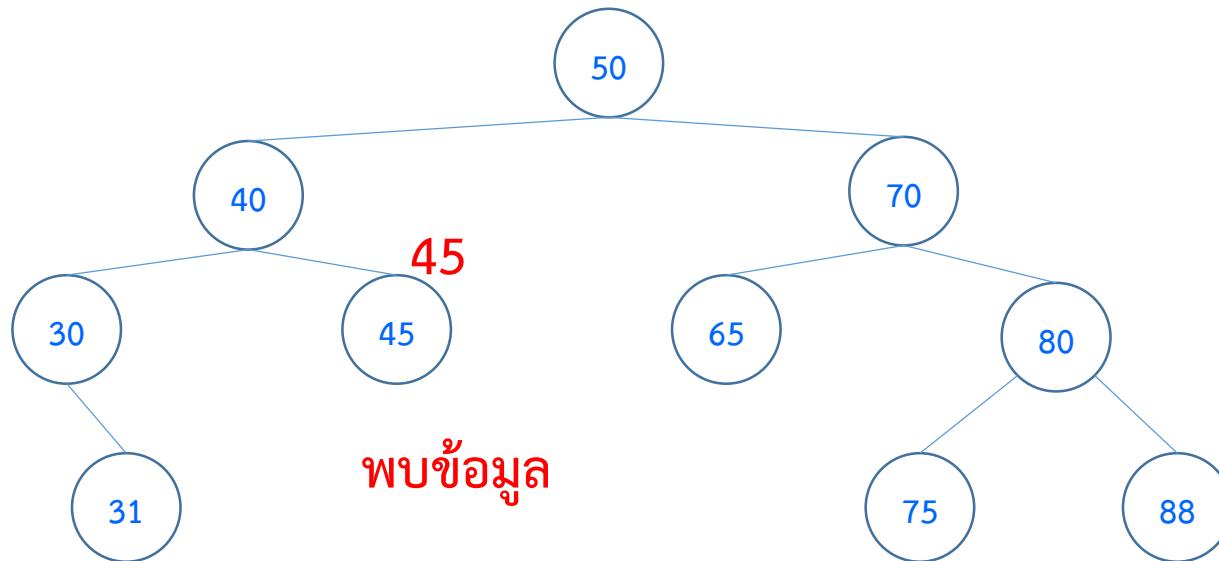
การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



Trees: Binary Search Trees

การค้นข้อมูลใน Binary Search Tree

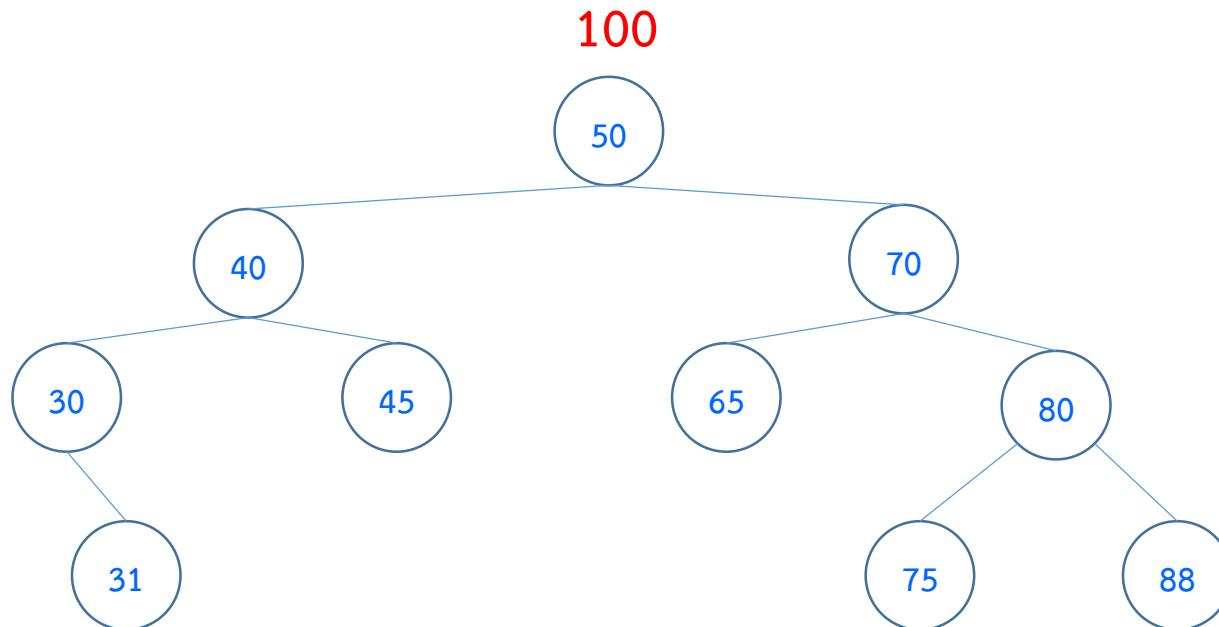
การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



Trees: Binary Search Trees

การค้นข้อมูลใน Binary Search Tree

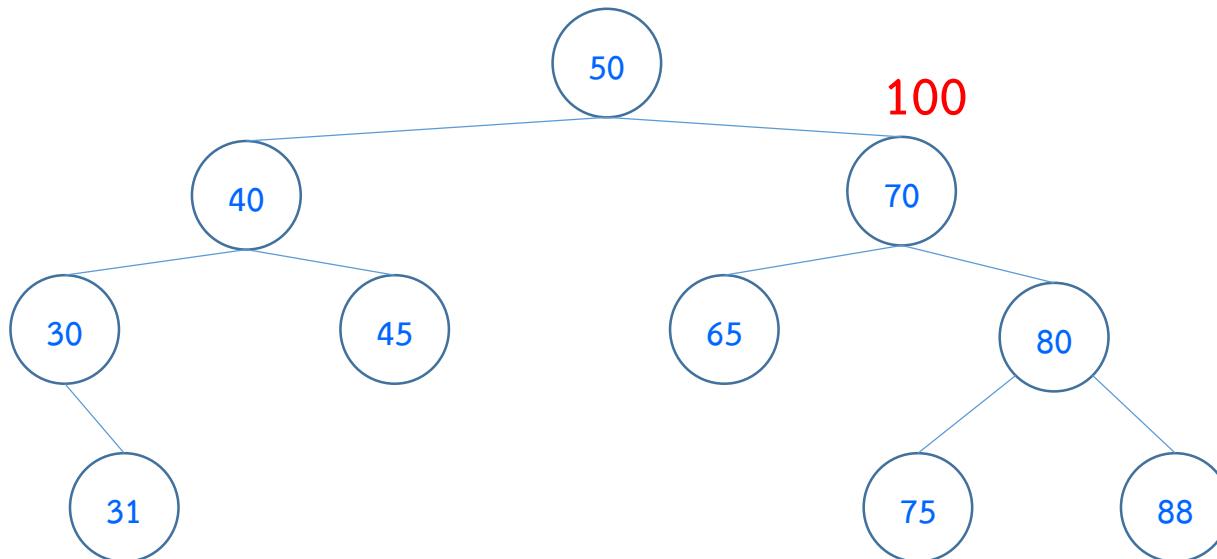
การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



Trees: Binary Search Trees

การค้นข้อมูลใน Binary Search Tree

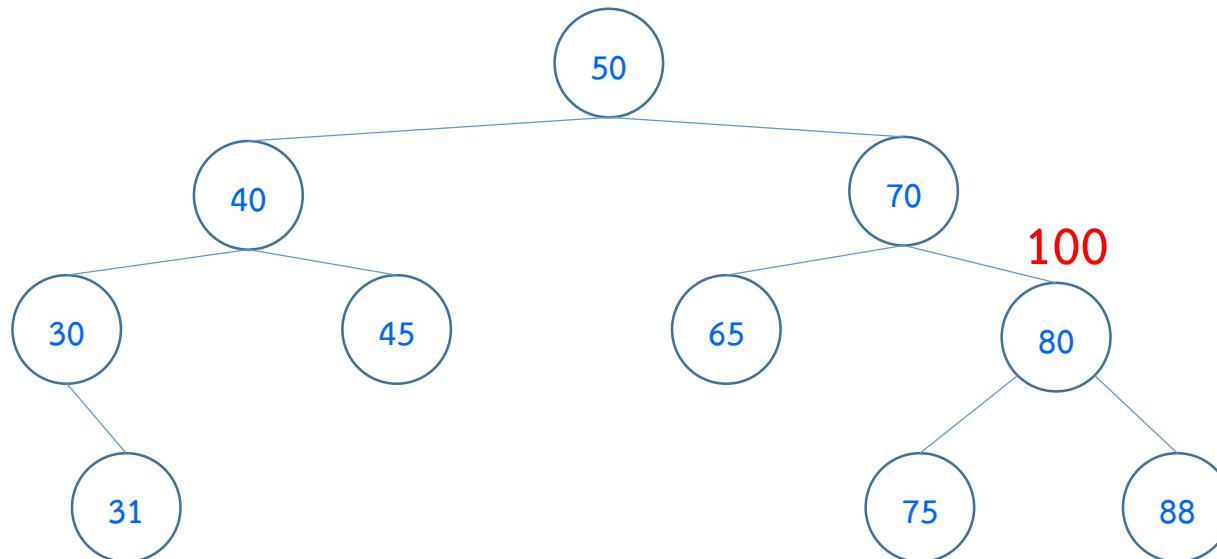
การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



Trees: Binary Search Trees

การค้นข้อมูลใน Binary Search Tree

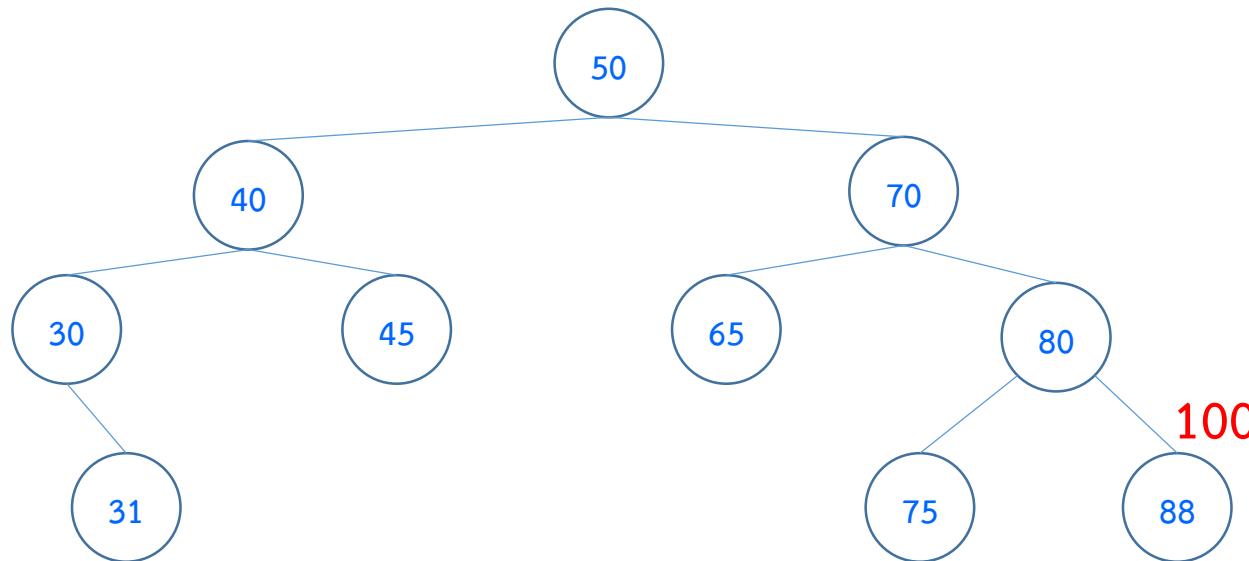
การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



Trees: Binary Search Trees

การค้นข้อมูลใน Binary Search Tree

การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf



Trees: Binary Search Trees

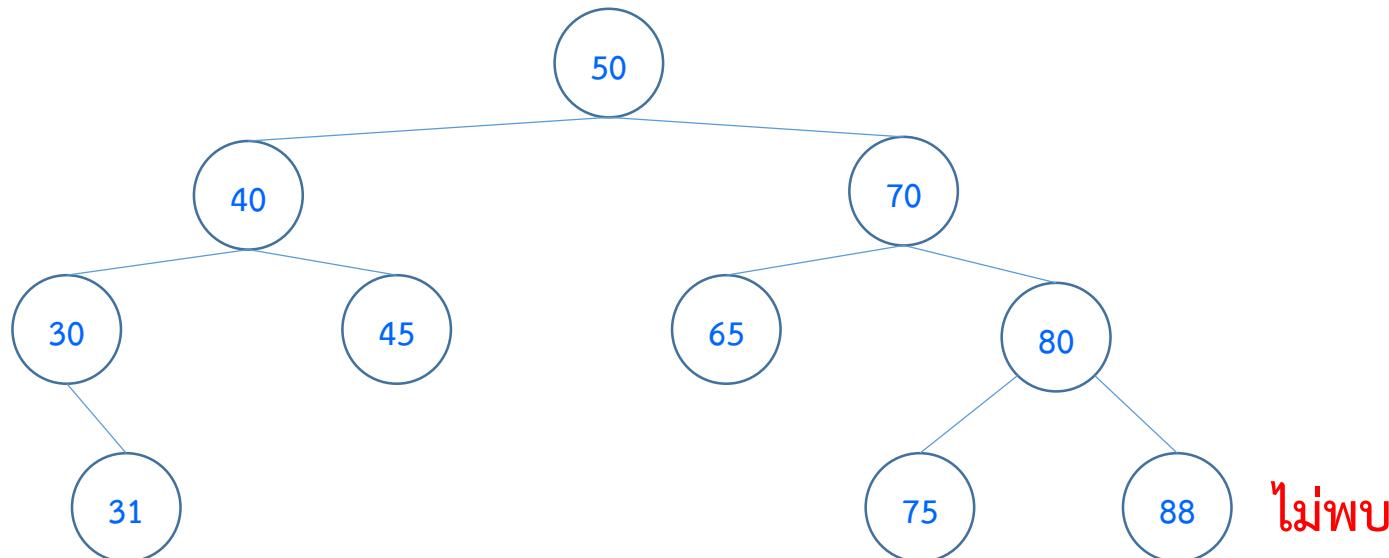
การค้นข้อมูลใน Binary Search Tree

การค้นข้อมูล ใช้สำหรับตรวจสอบว่าต้นไม้มีค่าที่ต้องการหาหรือไม่
หรือใช้ในการดึงข้อมูลออกมาจากต้นไม้โดยค้นจาก ID
วิธีการจะเหมือนกับการเพิ่ม node ใหม่ โดยเริ่มจาก root ถึง leaf

ในกรณีที่ต้นไม้เป็น Complete tree

การค้นข้อมูล ไม่จำเป็นต้องໄ�回บันทึก node

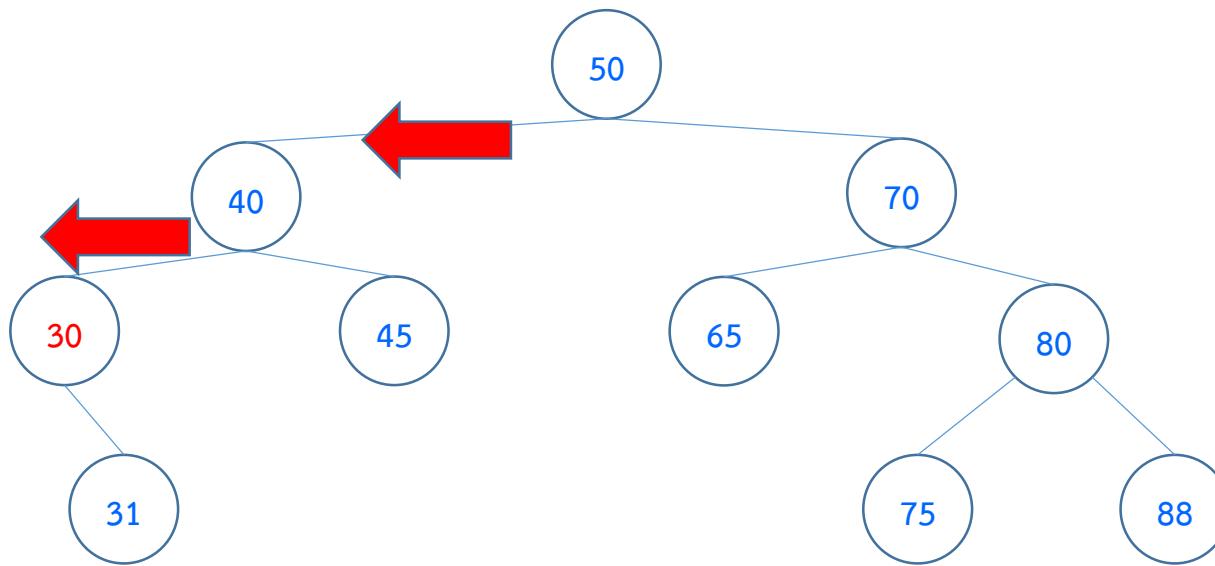
ทำให้การค้นข้อมูลทำได้เร็วกว่า Array และ Link list



Trees: Binary Search Trees

การหาค่าน้อยสุดใน Binary Search Tree

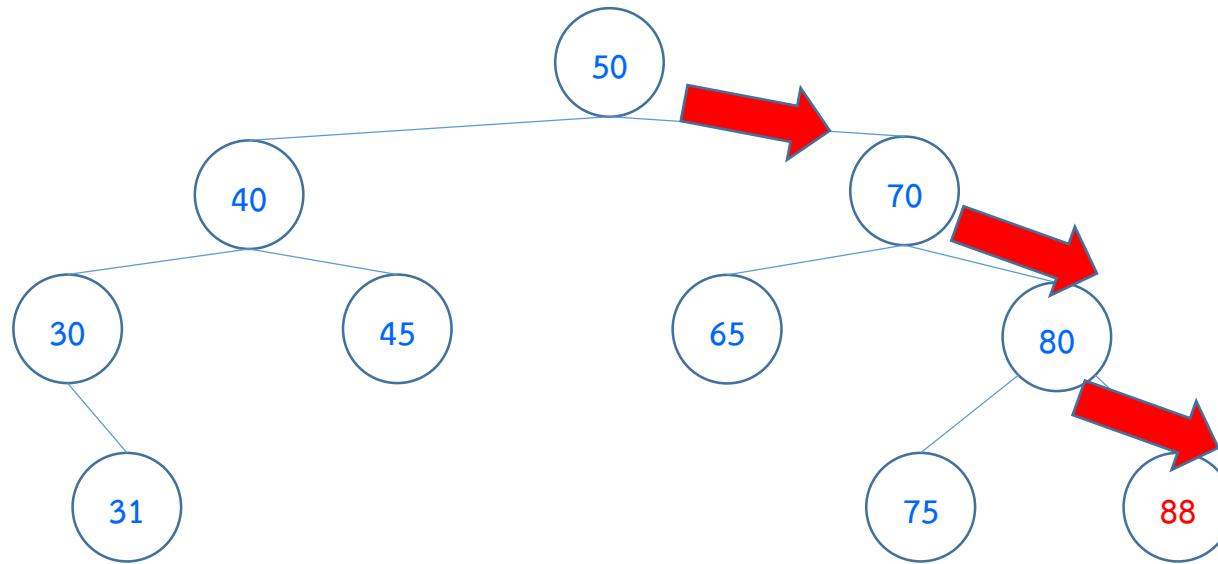
ให้เริ่มจาก Root และไถ่มาทางซ้ายตลอด จนกว่าจะไม่มีทางไป node สุดท้ายจะมีค่าน้อยสุดเสมอ



Trees: Binary Search Trees

การหาค่ามากสุดใน Binary Search Tree

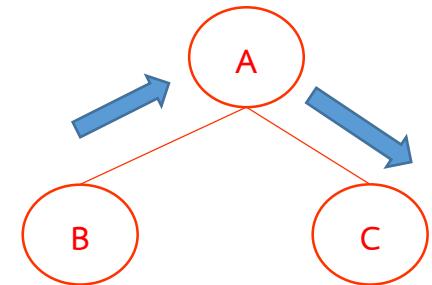
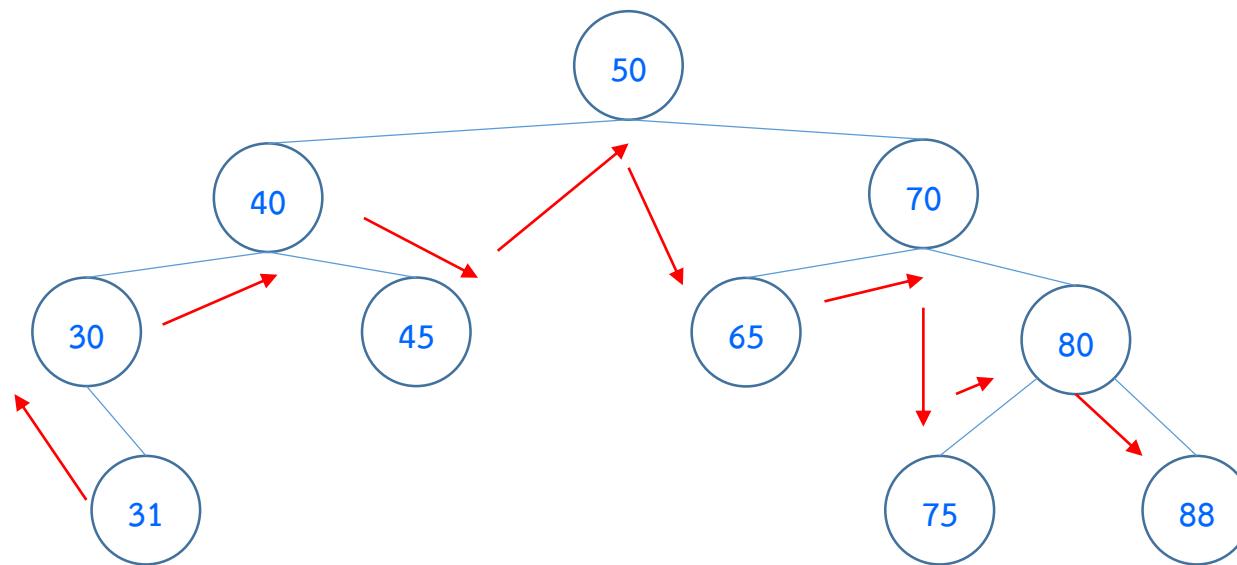
ให้เริ่มจาก Root และไถ่มาทางขวาตลอด จนกว่าจะไม่มีทางไป node สุดท้ายจะมีค่าน้อยสุดเสมอ



Trees: Binary Search Trees

การเรียงข้อมูลใน Binary Search Tree

ให้ทำการ Traverse แบบ Inorder จะได้ข้อมูลที่เรียงจากน้อยไปมาก เช่น



การท่องแบบ Inorder
B → A → C

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

ความเร็วในการทำงาน

การ Traverse ใช้เวลา $O(N)$

การ Add ใช้เวลาเท่าความสูงของต้นไม้คือ $O(\log_2 N)$ หากต้นไม้ลู่ไปทางเดียว จะใช้เวลา $O(N)$

การ ค้นข้อมูล ใช้เวลาเท่าความสูงของต้นไม้คือ $O(\log_2 N)$ หากต้นไม้ลู่ไปทางเดียว จะใช้เวลา $O(N)$

ต้นไม้ลู่ไปทางเดียว = Linked-list

ทำไมเราถึงใช้ต้นไม้ในการเก็บข้อมูล ?

ในการใช้งานจริง ต้นไม้มักจะเป็น Complete tree

Complete tree จะสูง $\log_2 N$

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

หากใช้ต้นไม้ จะใช้เวลา $O(\log_2 N)$

$$\log_2(1000000) = 19.93$$

หรือประมาณ 20 ครั้งเท่านั้น

Trees: Binary Search Trees

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