Chapter 6 Tree

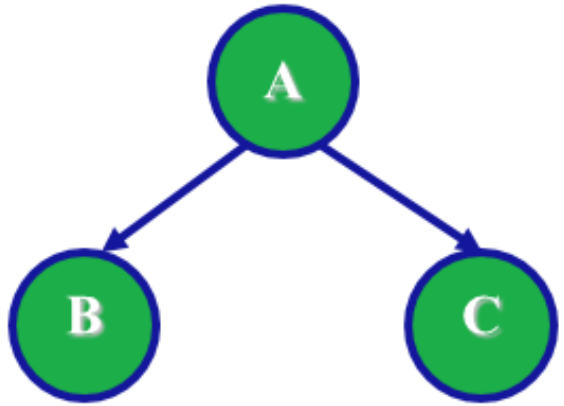
ผศ.ดร.สิลดา อินทรโสธรฉันท์

สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ มหาวิทยาลัยขอนแก่น

Tree Terminology

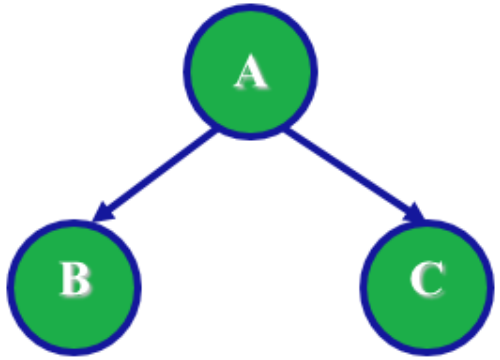
- Tree ประกอบด้วย Node และ Branch
- Node ทำหน้าที่ในการเก็บข้อมูล
- Branch ทำหน้าที่ในการเชื่อม node เข้าด้วยกัน



Node	**A, B, C**
Branch	**AB, AC**

Tree Terminology

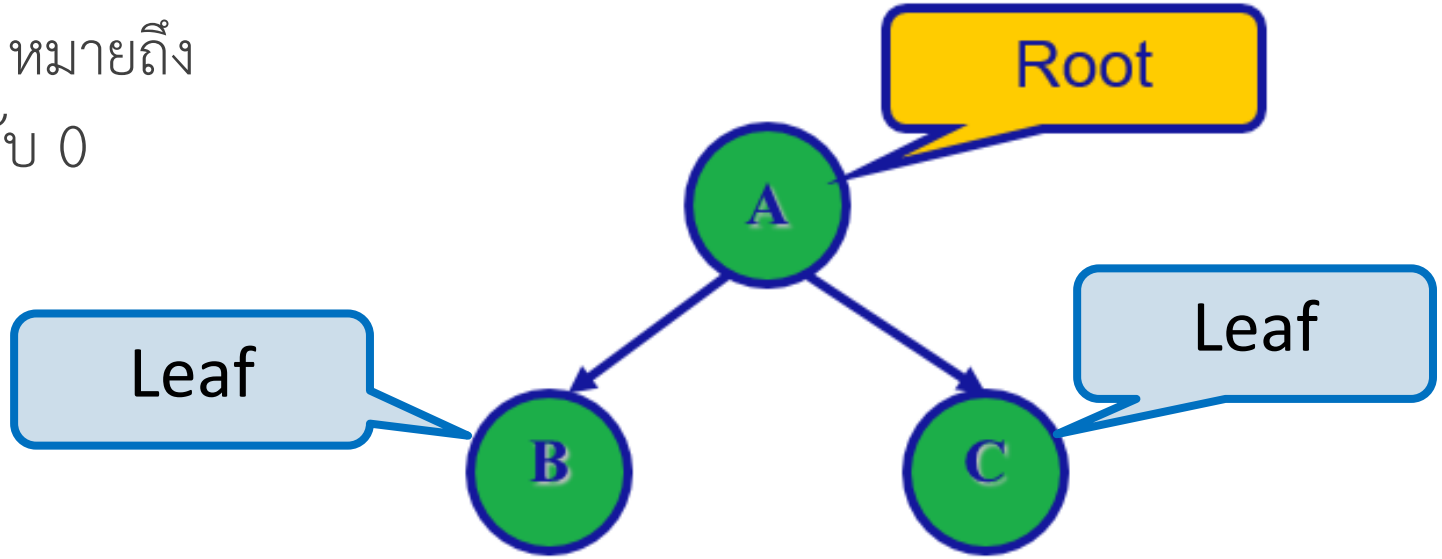
- Degree หมายถึง จำนวน Branch ที่สัมพันธ์กับ node แบ่งเป็น 2 ประเภท
- Indegree หมายถึง Branch ที่เข้าหา node
- Outdegree หมายถึง Branch ที่ออกจาก node



Node A มี Degree เท่ากับ 2
Indegree = 0
Outdegree = 2

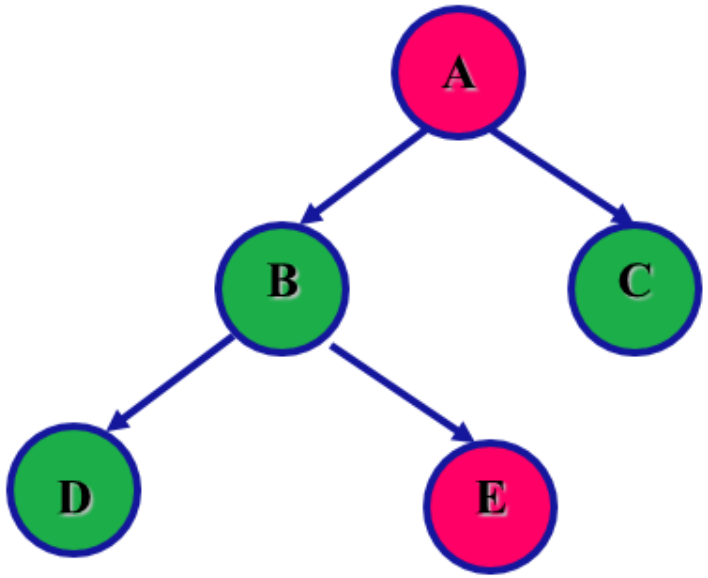
Tree Terminology

- Root หมายถึง node แรกของ Tree
- Leaf หรือ External node หมายถึง node ที่มี Outdegree เท่ากับ 0



Tree Terminology

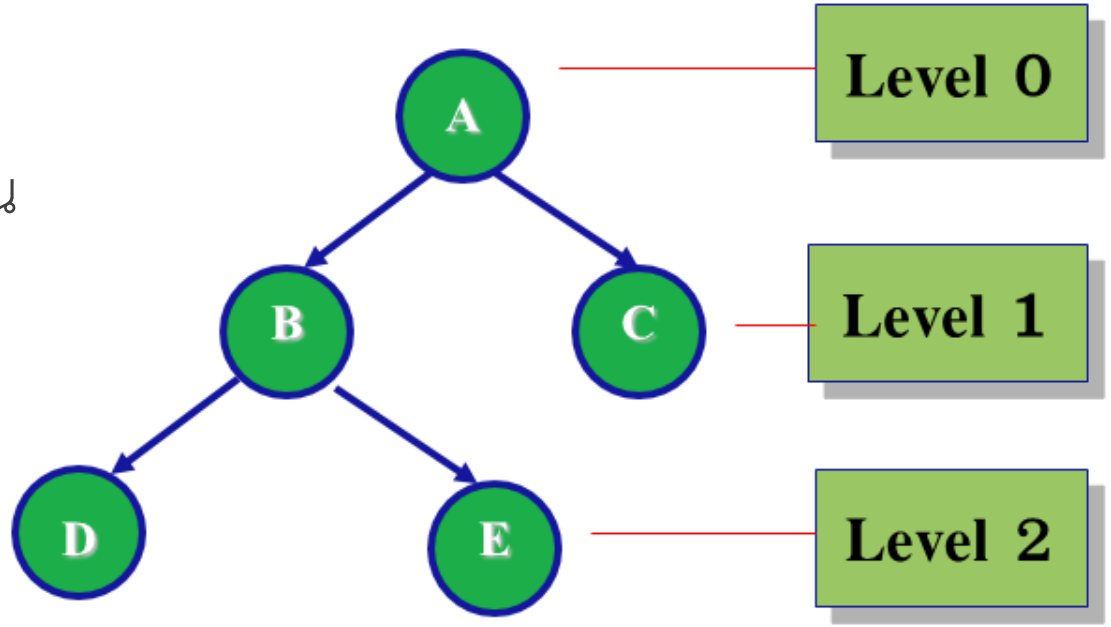
- Path หมายถึง เส้นทางจาก node หนึ่งไปยังอีก node หนึ่ง
- ทุก node ใน Tree จะต้องมี Path เดียวเท่านั้น



Path จาก A ไป E **A -> B -> E**

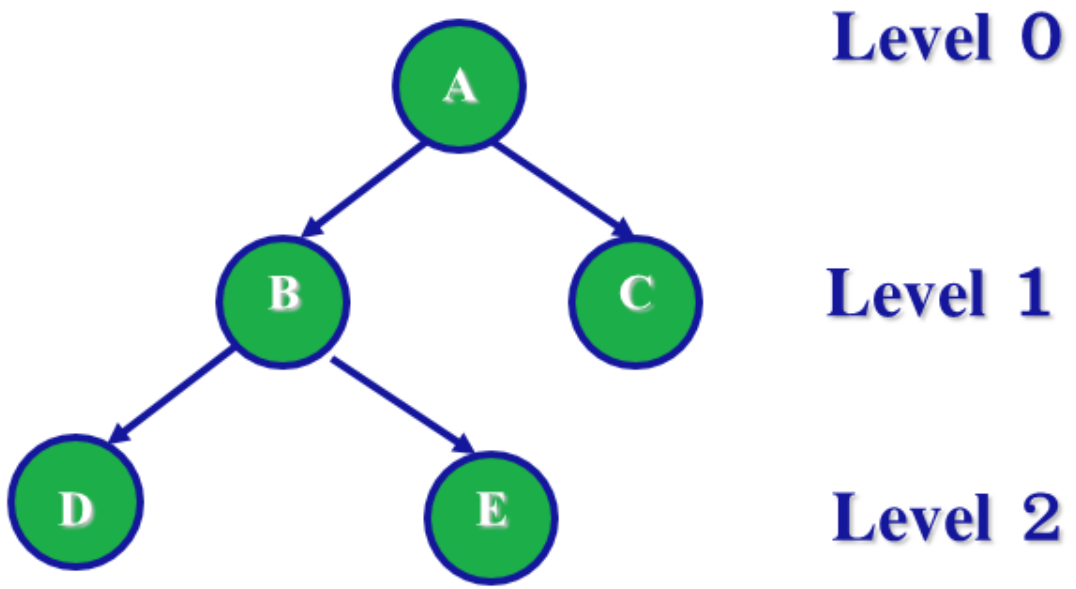
Tree Terminology

- Level หมายถึง ระยะทางจาก Root
- Depth ของ node หมายถึง ความยาวของ path จาก root node ถึง node นั้น ดังนั้น root node จึงมีความลึกเป็น 0



Tree Terminology

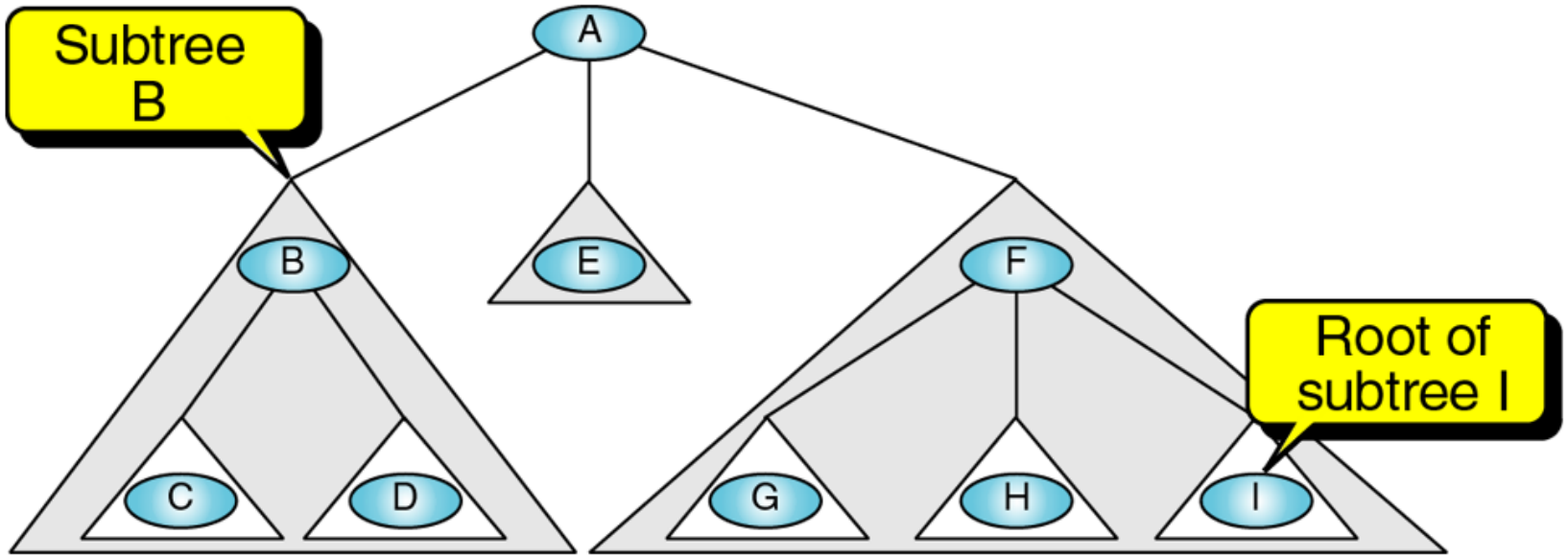
- Height ของ Tree หมายถึง Level สูงสุดของ Leaf บวกด้วย 1
- Depth ของ Tree หมายถึง ความลึกของ Leaf node ที่อยู่ลึกที่สุด ซึ่งจะมีค่าเท่ากับความสูงของ Tree เสมอ



Height = 2 + 1 = 3

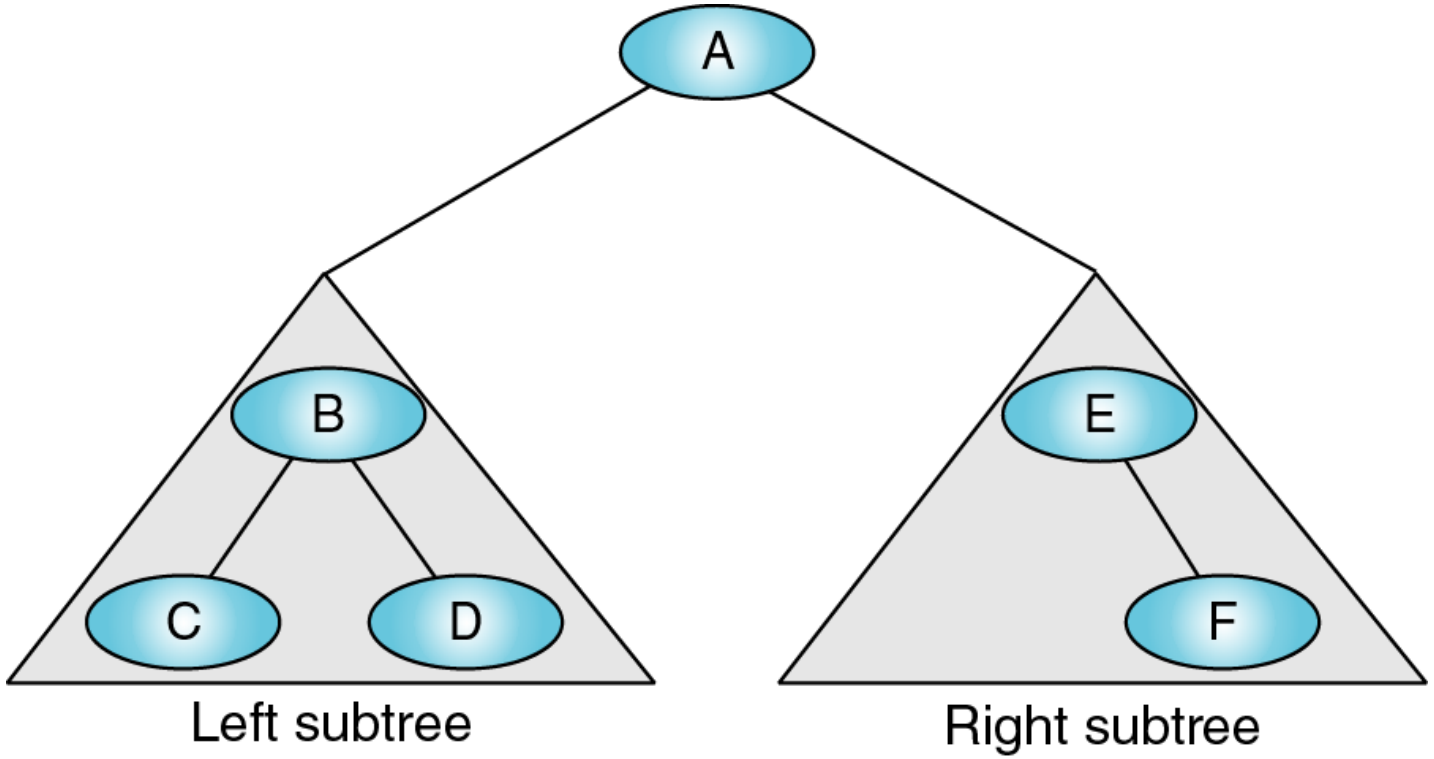
Tree Terminology

- Subtree หมายถึง โครงสร้างที่เชื่อมต่อกันภายใต้ Root โดย node แรกของ Subtree จะเป็น Root ของ Subtree นั้น และใช้เป็นชื่อเรียก Subtree
- Subtree สามารถแบ่งย่อยเป็น Subtree ได้อีกจนกว่าจะ Empty

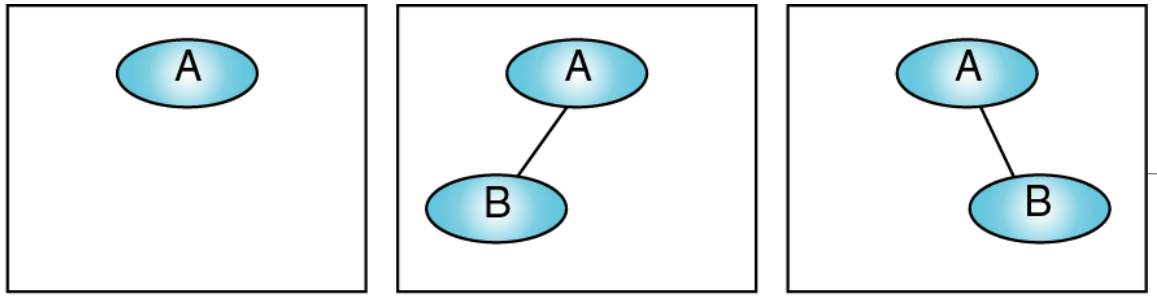


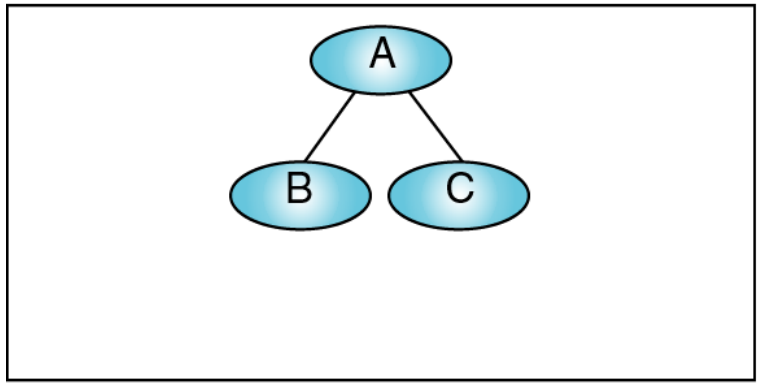
Tree Terminology

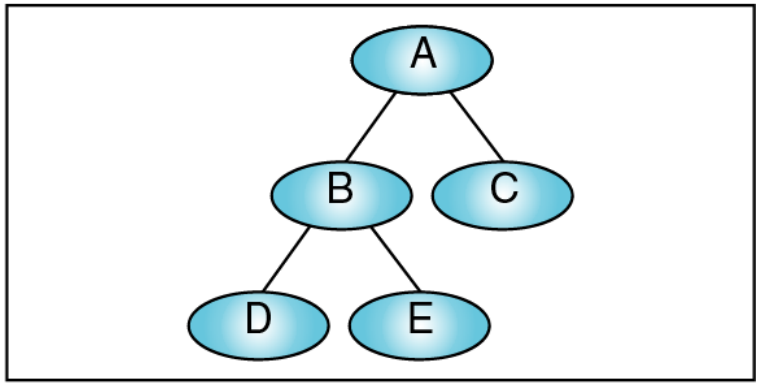
- **Binary Tree** หมายถึง ต้นไม้ที่แต่ละ node มี Outdegree ไม่เกิน 2

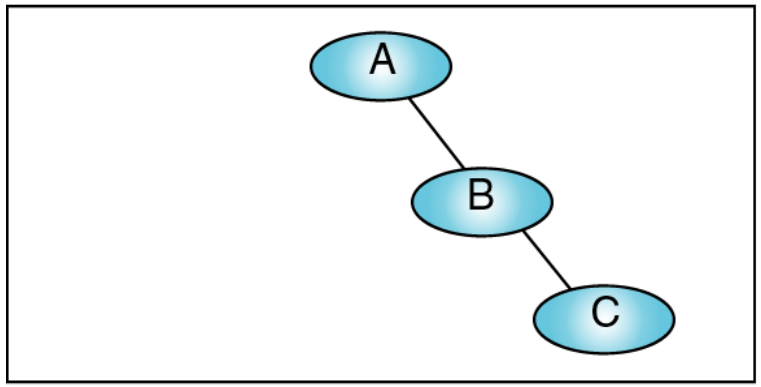


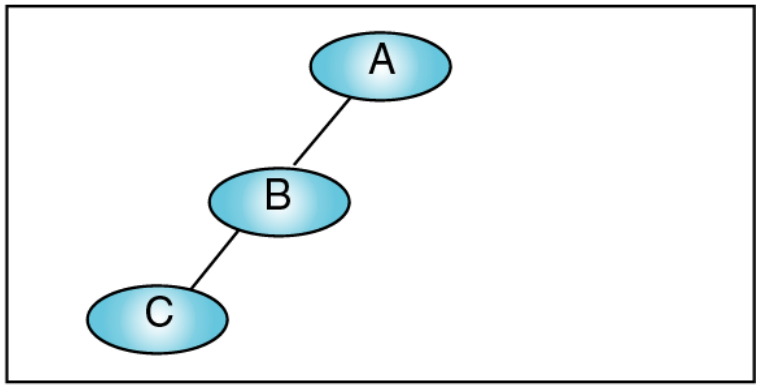
Binary Tree





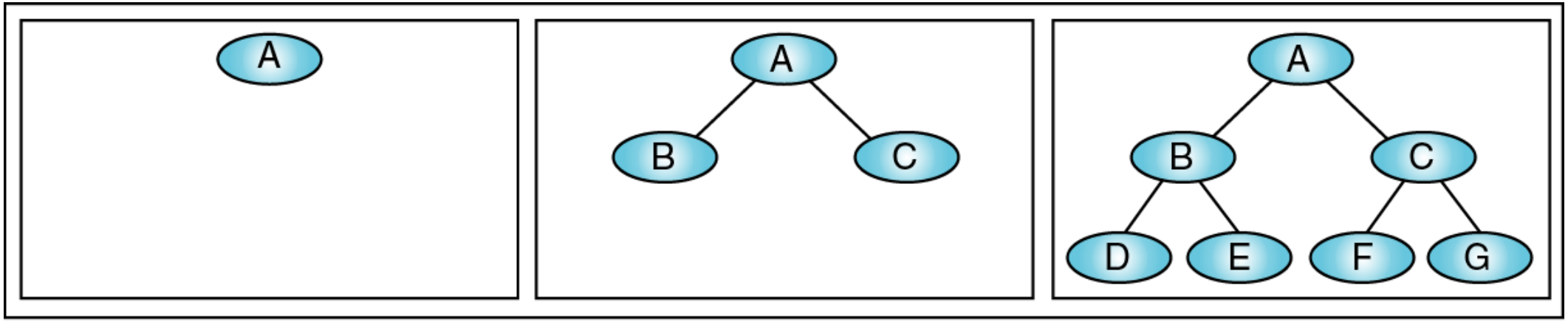






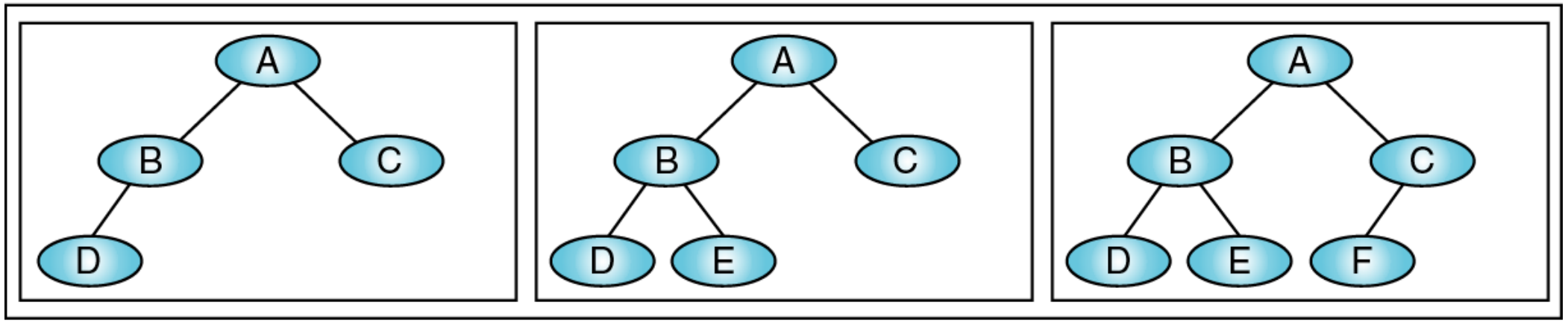
Tree Terminology

- **Complete Binary Tree** เป็น Binary tree ที่มี node เต็มทุก Level



Tree Terminology

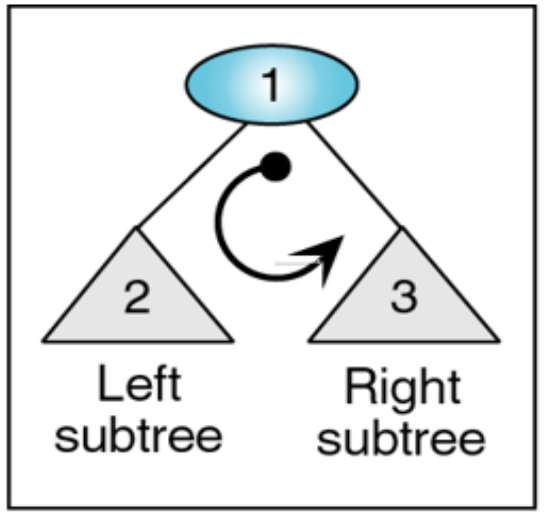
- **Nearly Complete Binary Tree** เป็น Binary Tree ที่มี node เต็มทุก Level ยกเว้น Level สุดท้าย และ Node ใน Level สุดท้ายอยู่เรียงกันทางซ้าย



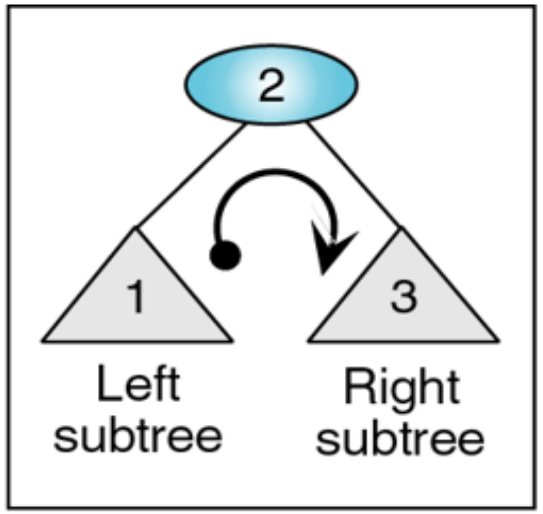
Binary Tree Traversal

- **Depth – First** : node ลูกทั้งหมดของ Child จะต้องถูกประมวลผลก่อน Child ถัดไป
- **Breath – First :** ประมวลผลทีละ Level จากบนลงล่าง

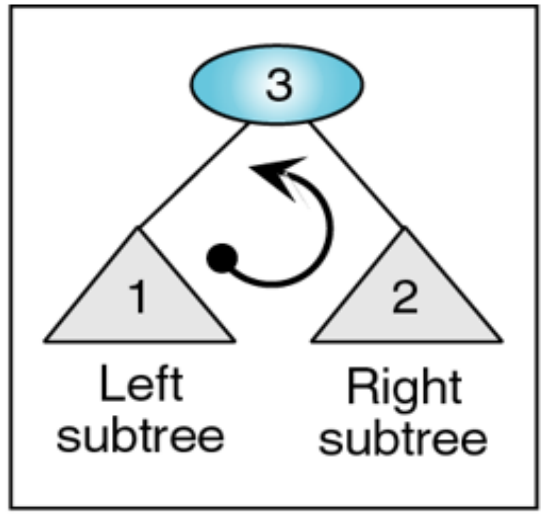
Depth – First



(a) Preorder traversal



(b) Inorder traversal



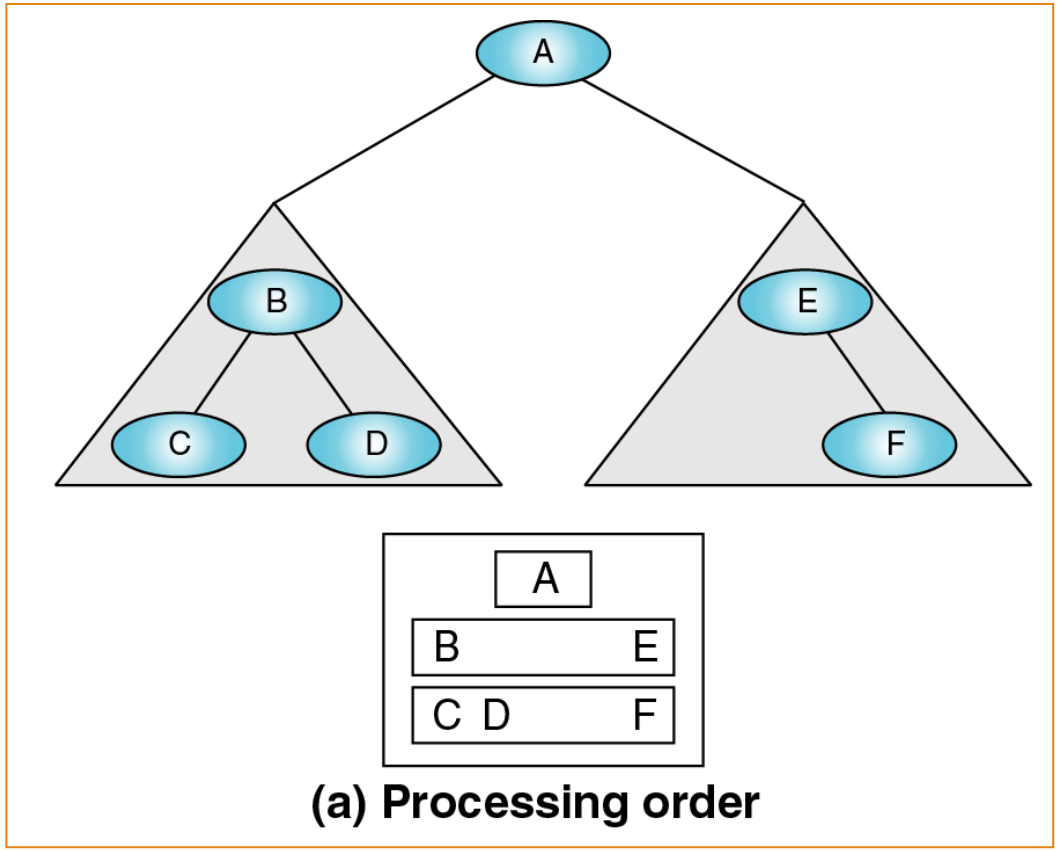
(c) Postorder traversal

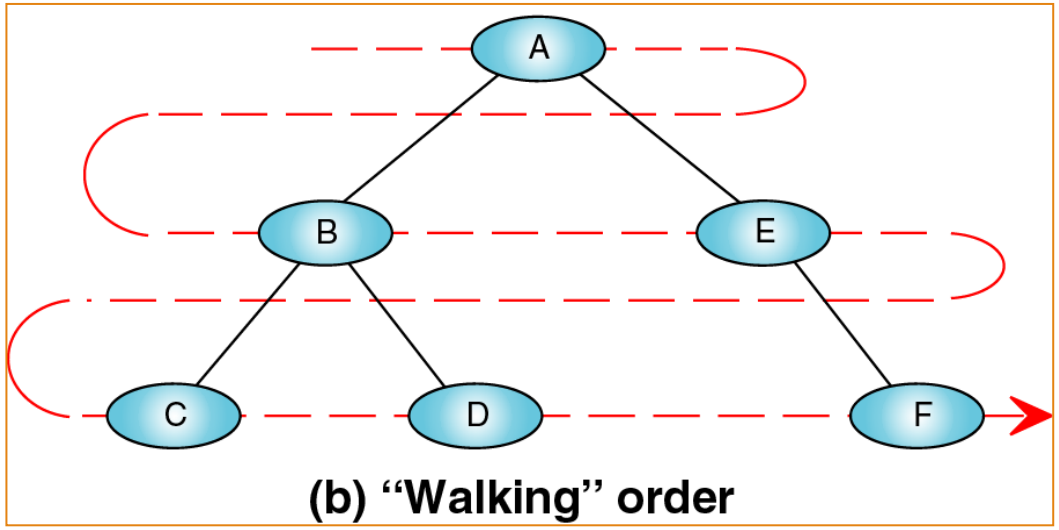
N L R

L N R

L R N

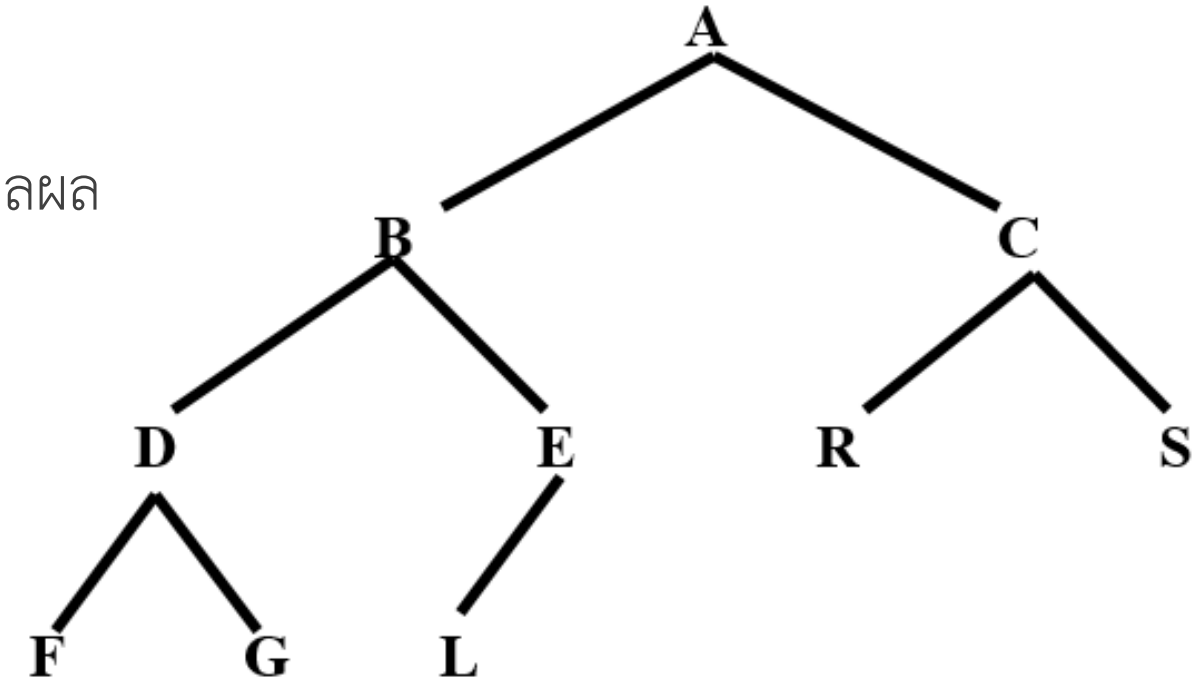
Breath – First





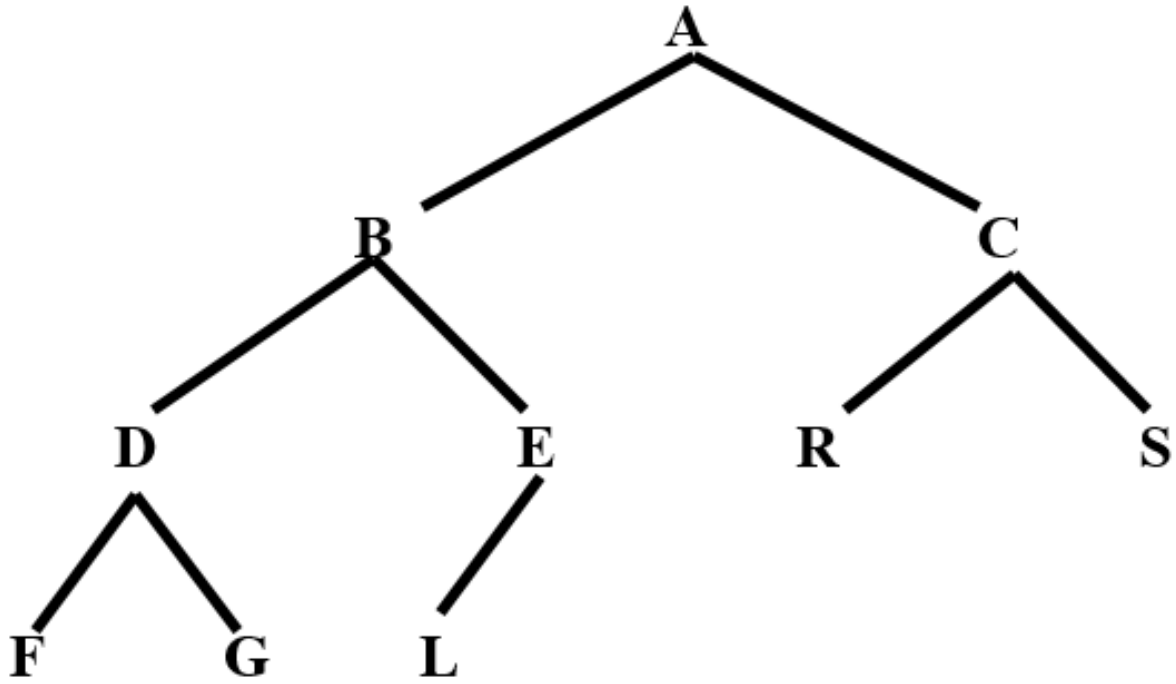
ตัวอย่าง

- จากต้นไม้ต่อไปนี้ จงหาลำดับการประมวลผลแบบ Depth – First ทั้งหมด
- จากต้นไม้ต่อไปนี้ จงหาลำดับการประมวลผลแบบ Breath – First ทั้งหมด



ตัวอย่าง

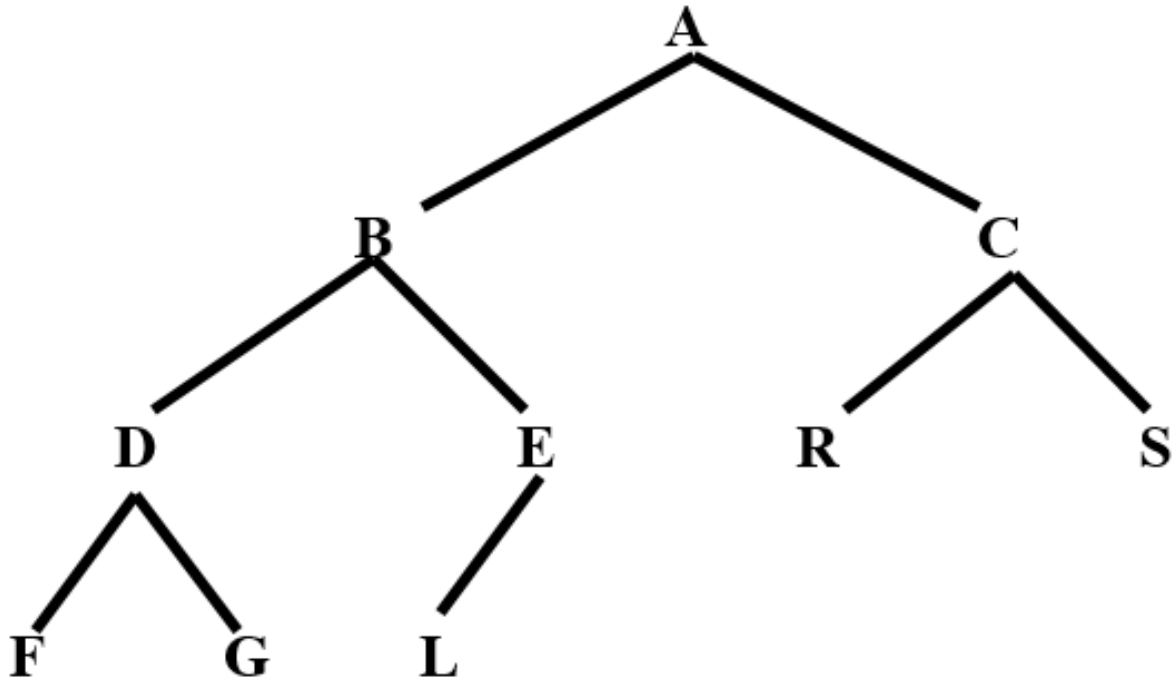
- จากต้นไม้ต่อไปนี้ จงหาลำดับการประมวลผลแบบ Depth – First ทั้งหมด
- preorder : A B D F G E L C R S
- inorder : F D G B L E A R C S
- Postorder : F G D L E B R S C A



ตัวอย่าง

- จากต้นไม้ต่อไปนี้ จงหาลำดับการประมวลผลแบบ Breath – First ทั้งหมด

>> A B C D E R S F G L



Preorder

algorithm preOrder (val root <node pointer>)

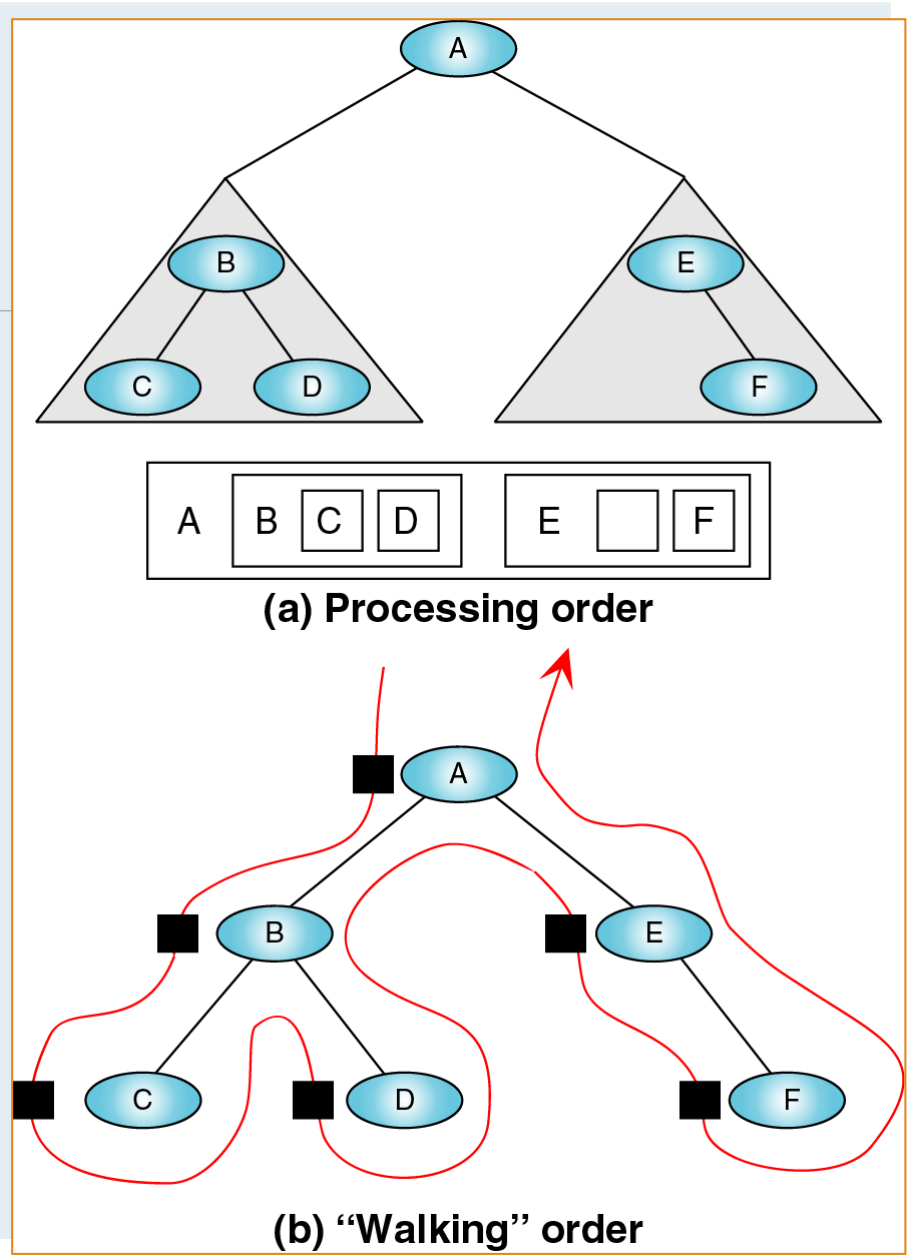
1 if (root is not null)

  1.1 process(root)

  1.2 preOrder(root->leftSubtree)

  1.3 preOrder(root->rightSubtree)

2 return



Inorder

algorithm inOrder (val root <node pointer>)

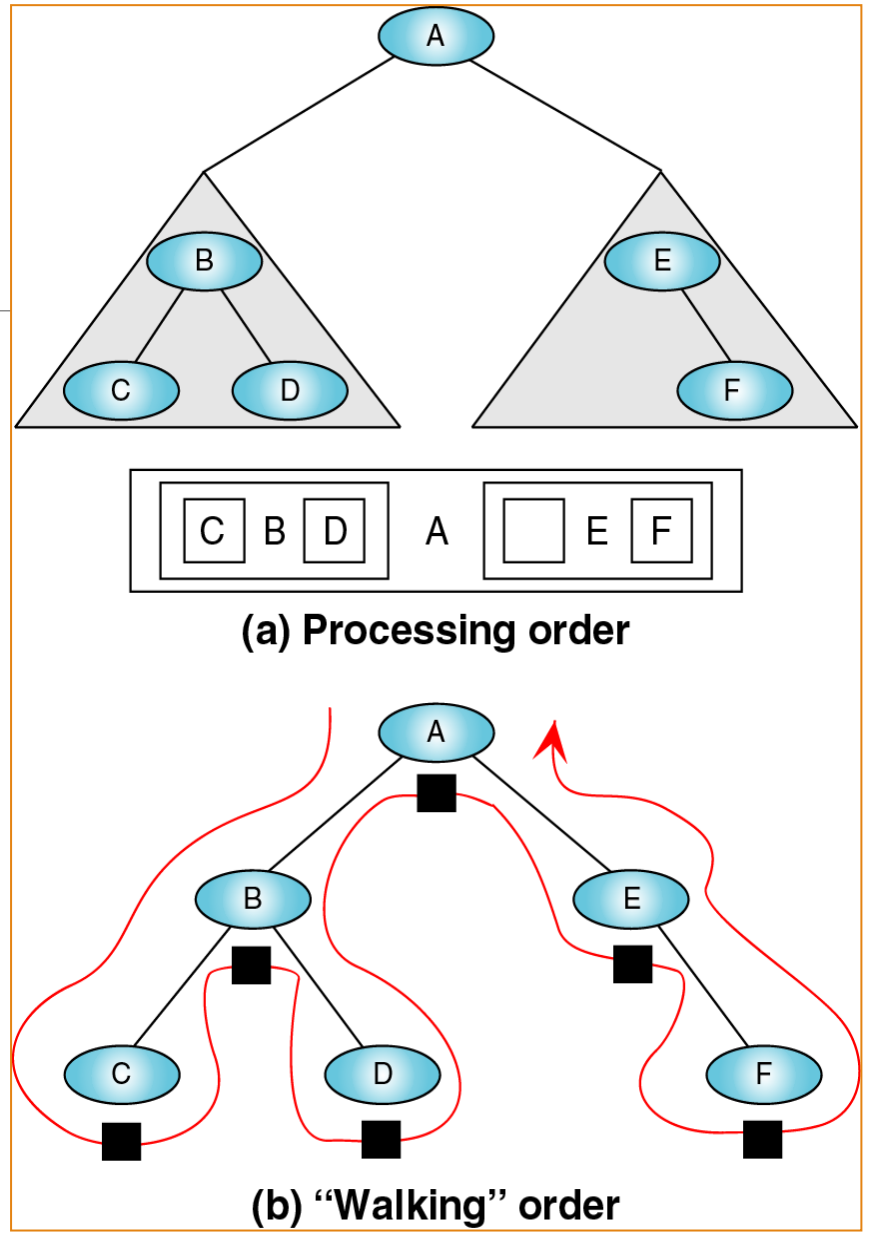
1 if (root is not null)

  1.1 inOrder(root->leftSubtree)

  1.2 process(root)

  1.3 inOrder(root->rightSubtree)

2 return



Postorder

algorithm postOrder (val root <node pointer>)

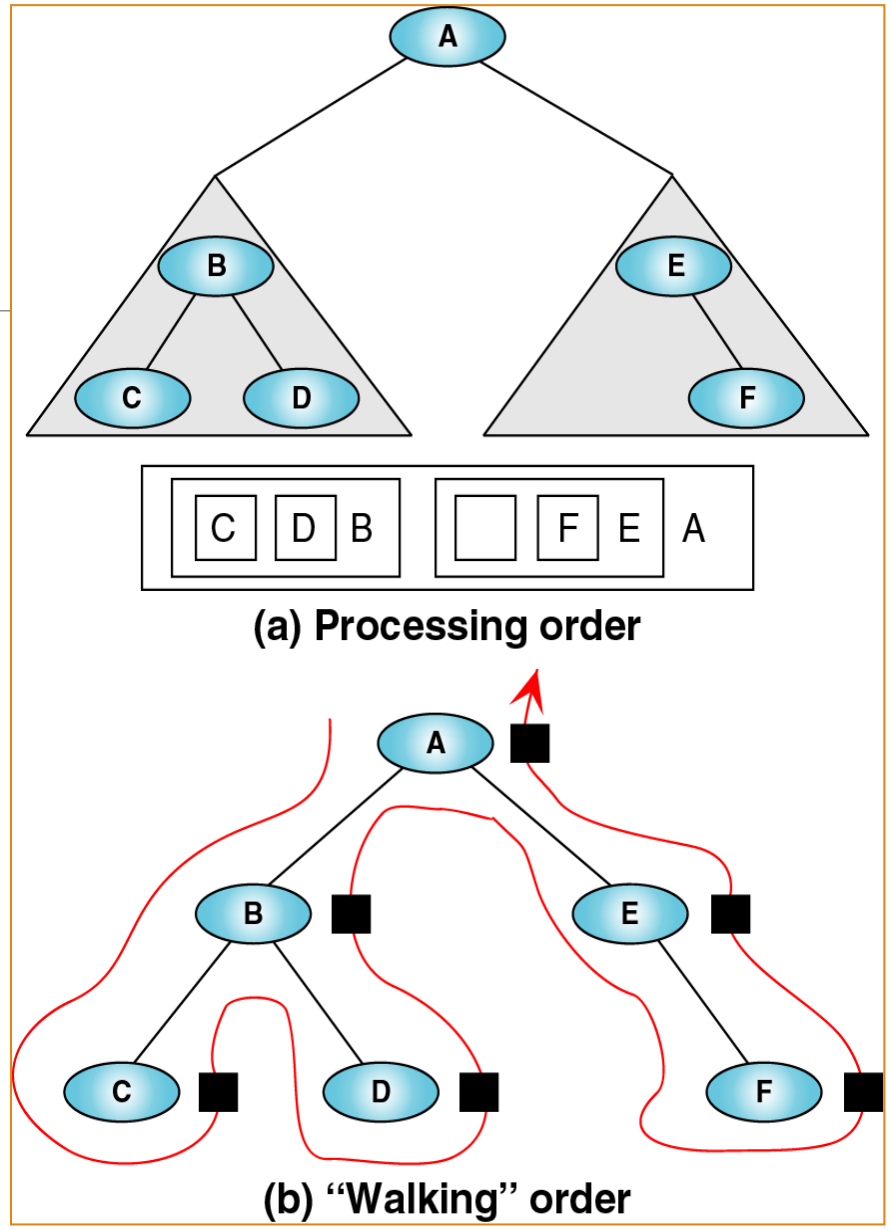
1 if (root is not null)

  1.1 postOrder(root->leftSubtree)

  1.2 postOrder(root->rightSubtree)

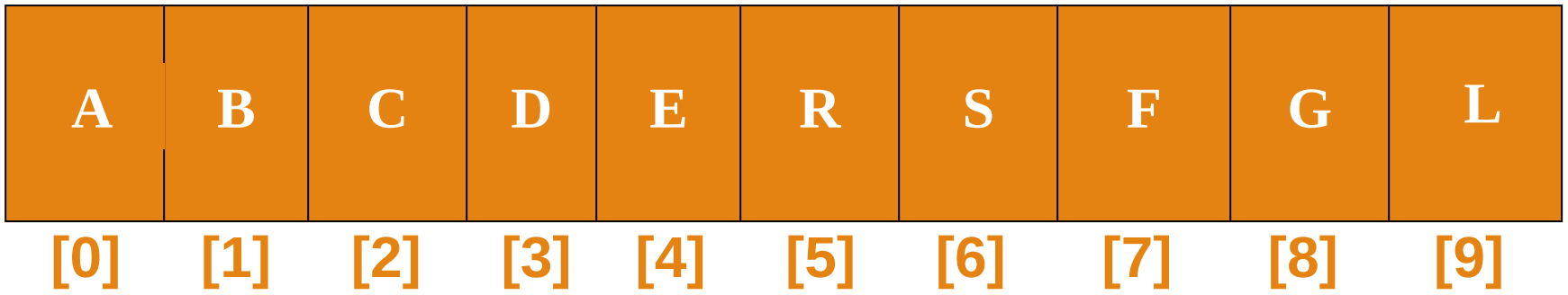
  1.3 process(root)

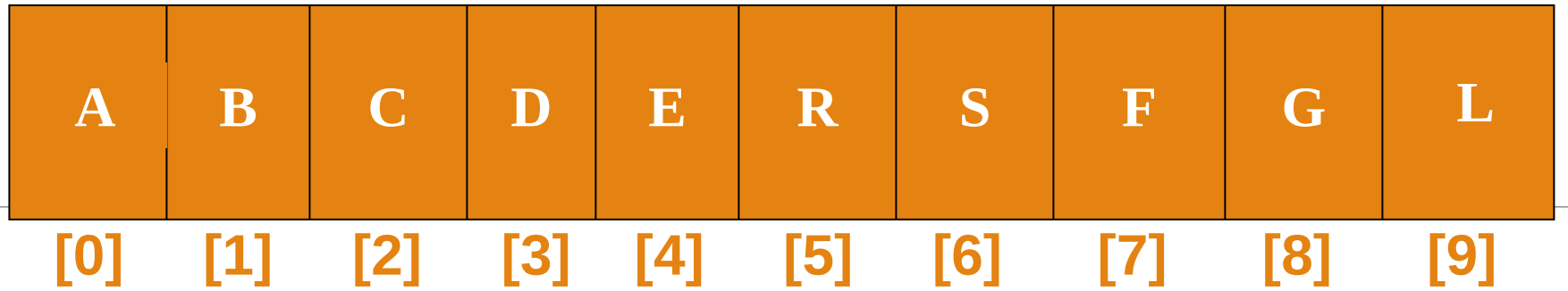
2 return



Implemented Complete Binary Tree by Array

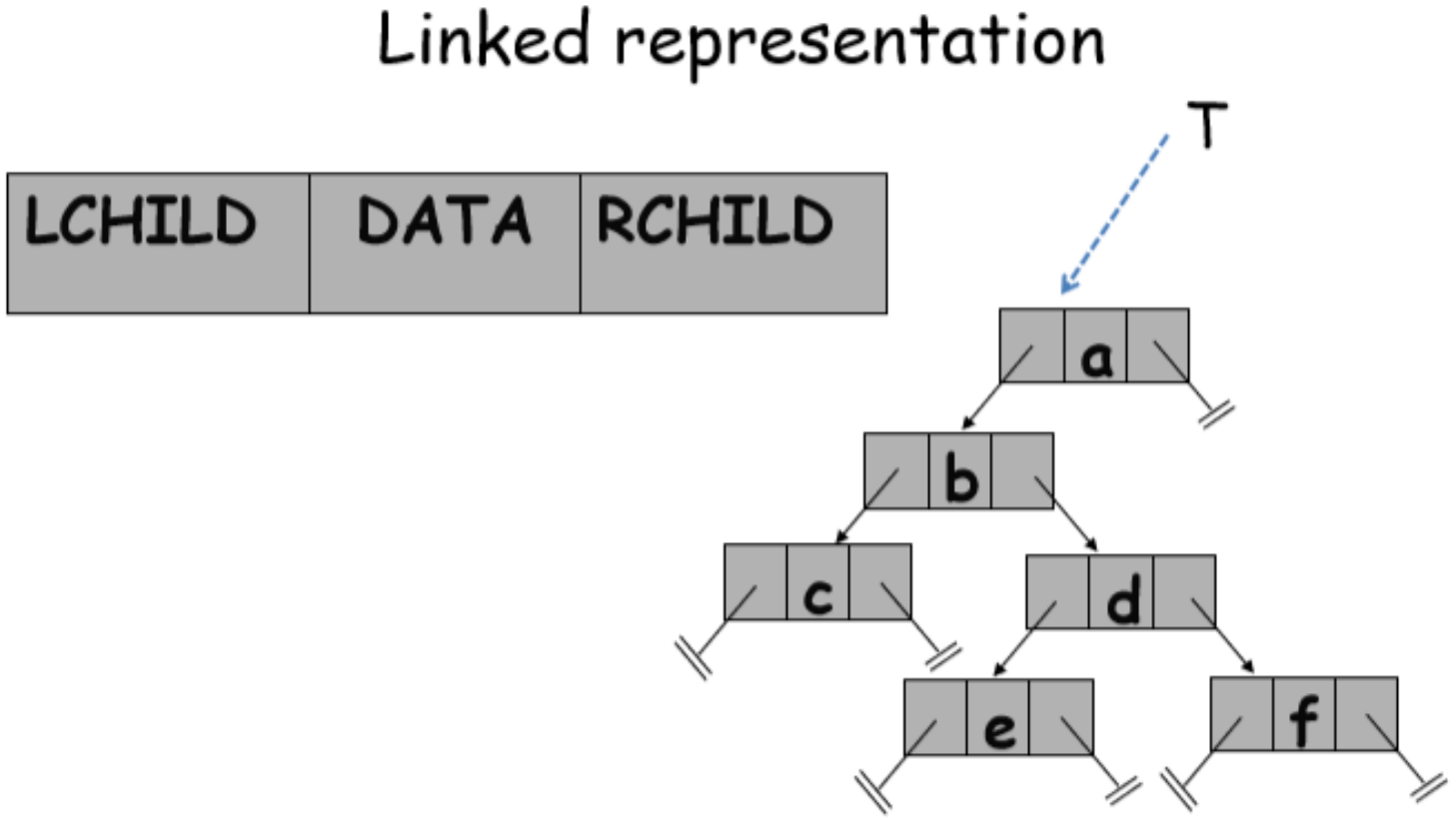
Parent at 0, children at 1, 2
Parent at 1, children at 3, 4
Parent at 2, children at 5, 6
…
Parent at i, children at ?





Child at 1, parent at 0
Child at 2, parent at 0
Child at 3, parent at 1
Child at 4, parent at 1
Child at 5, parent at 2
Child at 6, parent at 2
…
Child at i, parent at ?

Representation of Binary Trees



Implemented Binary Tree by Double Linked List

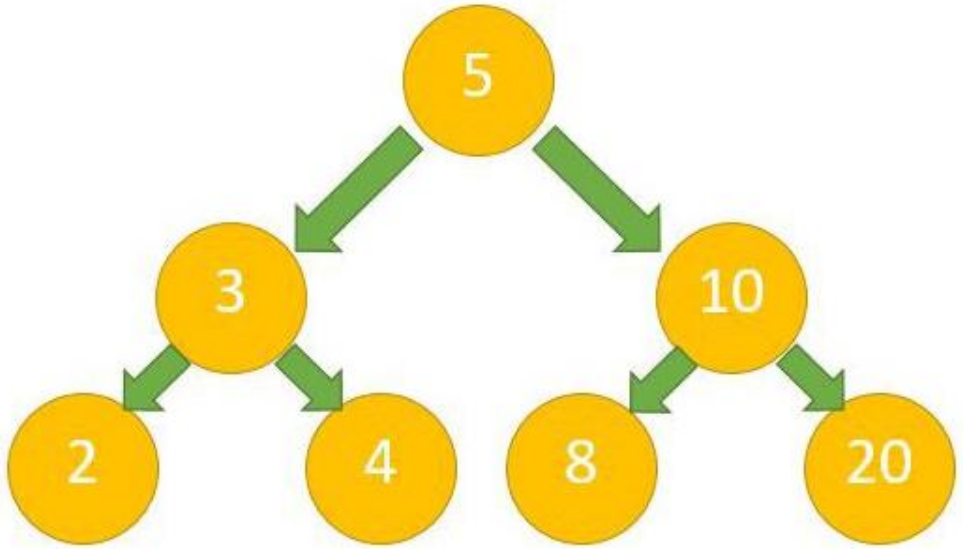
มีหลักการในการเก็บข้อมูลลง Tree ดังนี้

- Root คือ ข้อมูลตัวแรกที่อ่านเข้ามา
- ข้อมูลตัวถัดมา หากมีค่ามากกว่า Root จะเป็นลูกที่ฝั่งขวา หากมีค่าน้อยกว่า Root จะเป็นลูกที่ฝั่งซ้าย
- หากตำแหน่งดังกล่าวมีค่าอยู่แล้ว จะทำการตรวจสอบซ้ำว่าจะนำใส่ลงเป็นลูกฝั่งขวา หรือซ้ายต่อไป

Implemented Binary Tree by Double Linked List

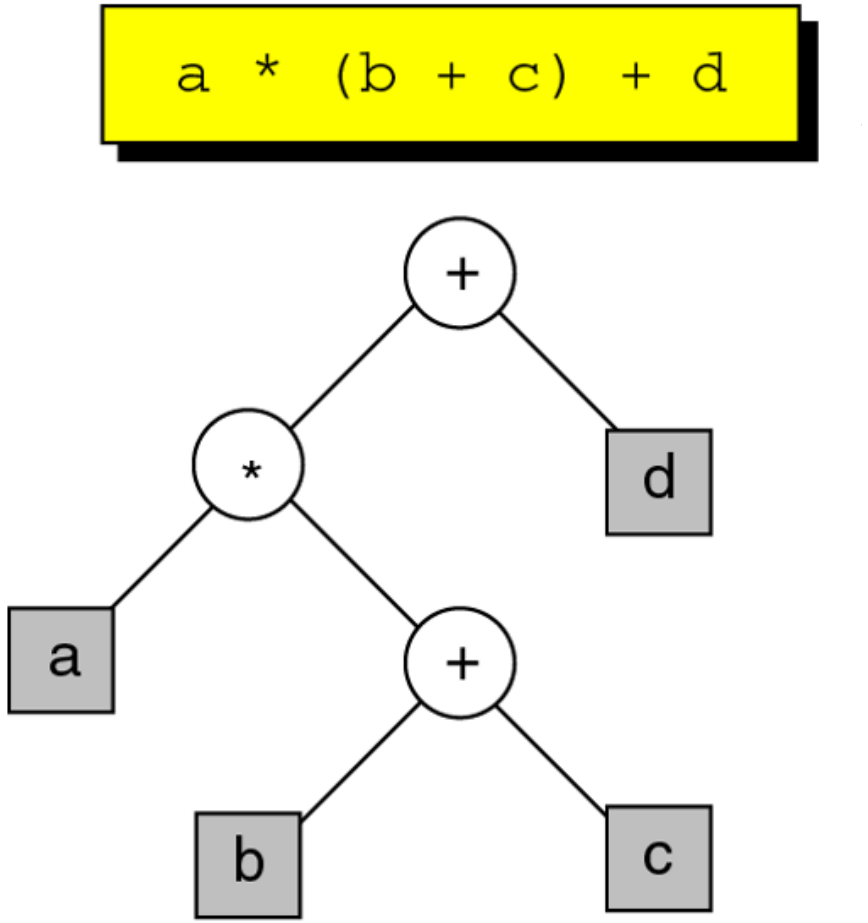
ชุดข้อมูล :

5 10 20 3 4 8 2



การประยุกต์ใช้ Tree

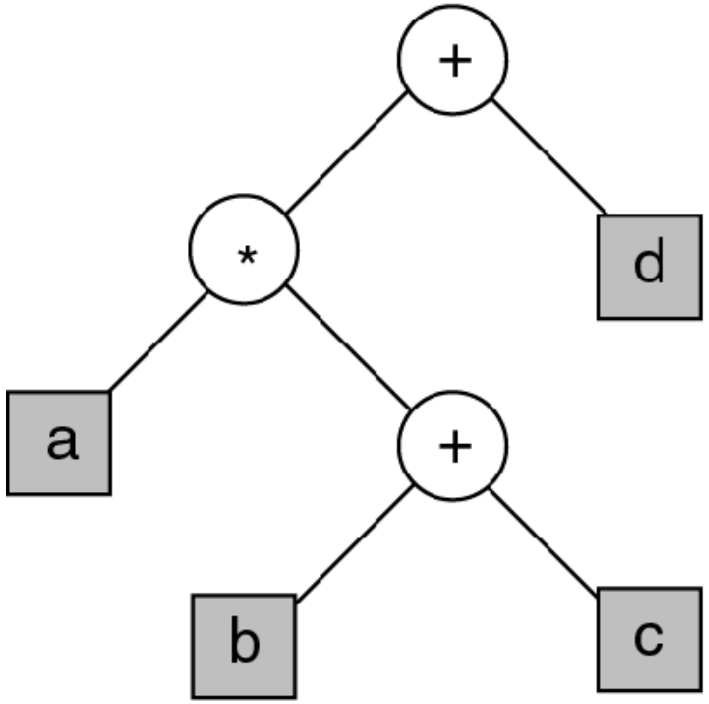
- Expression Tree หมายถึง Binary Tree ที่มีคุณสมบัติดังต่อไปนี้
  - Leaf เก็บ Operand
  - Root และ Internal node เก็บ Operator
  - Subtree เป็น Sub expression



การท่องใน Expression Tree

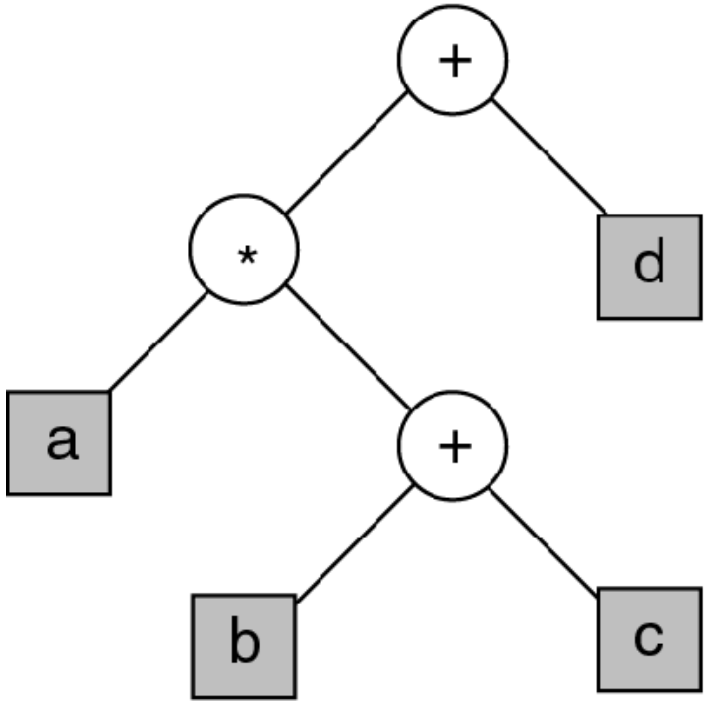
- Preorder Traversal  Prefix Expression
- Postorder Traversal  Postfix Expression
- Inorder Traversal  Infix Expression

Preorder Traversal



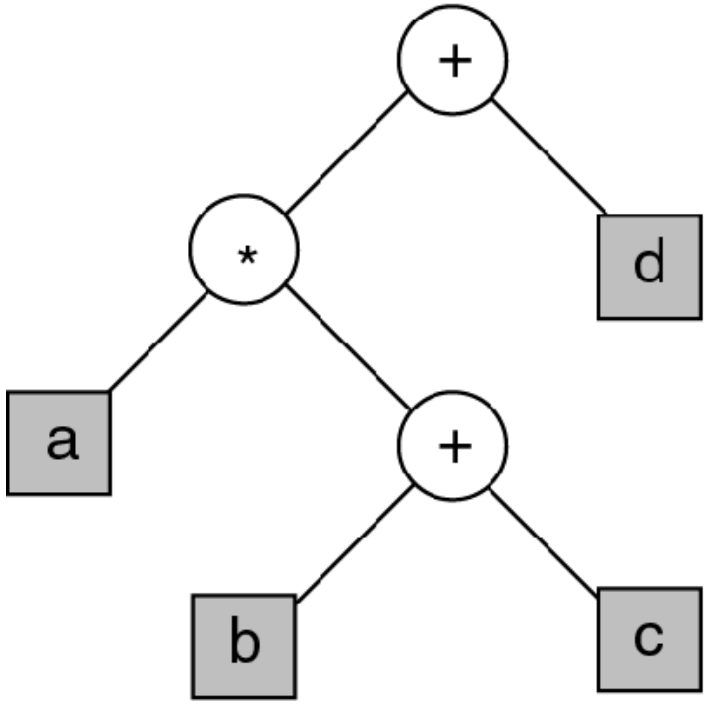
+ * a + b c d

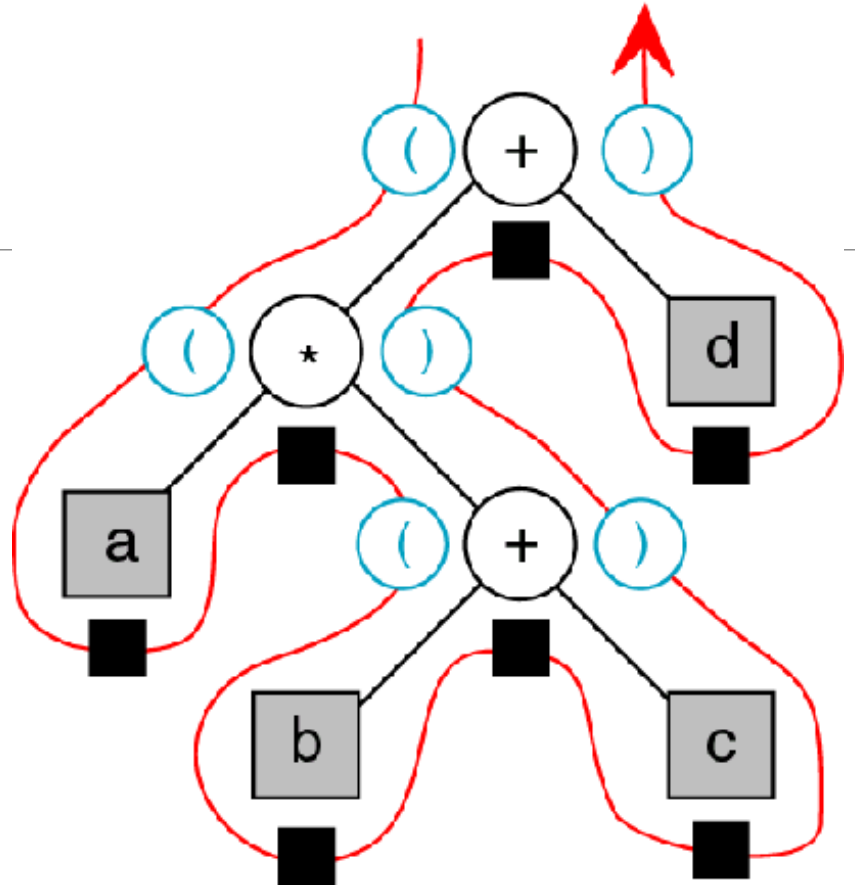
Postorder Traversal



a b c + * d +

Inorder Traversal



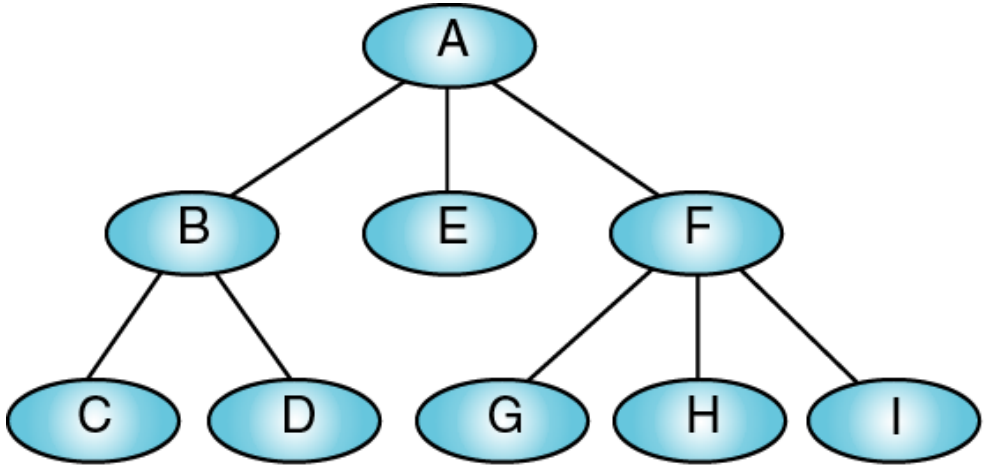


((a * (b + c)) + d)

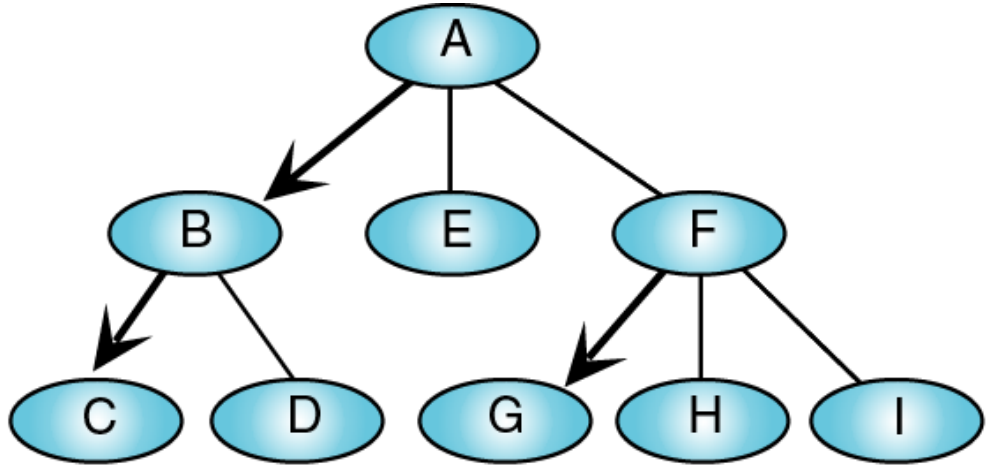
General Tree

- General Tree หมายถึง Tree ที่สามารถมี Outdegree ได้ไม่จำกัดจำนวน
- การแปลง General Tree เป็น Binary Tree มี 3 ขั้นตอน
  - ระบุ Child ที่อยู่ทางซ้ายสุด
  - เชื่อม Sibling เข้าด้วยกัน
  - ลบ Branch ที่ไม่ต้องการ

ตัวอย่างการแปลง General Tree เป็น Binary Tree

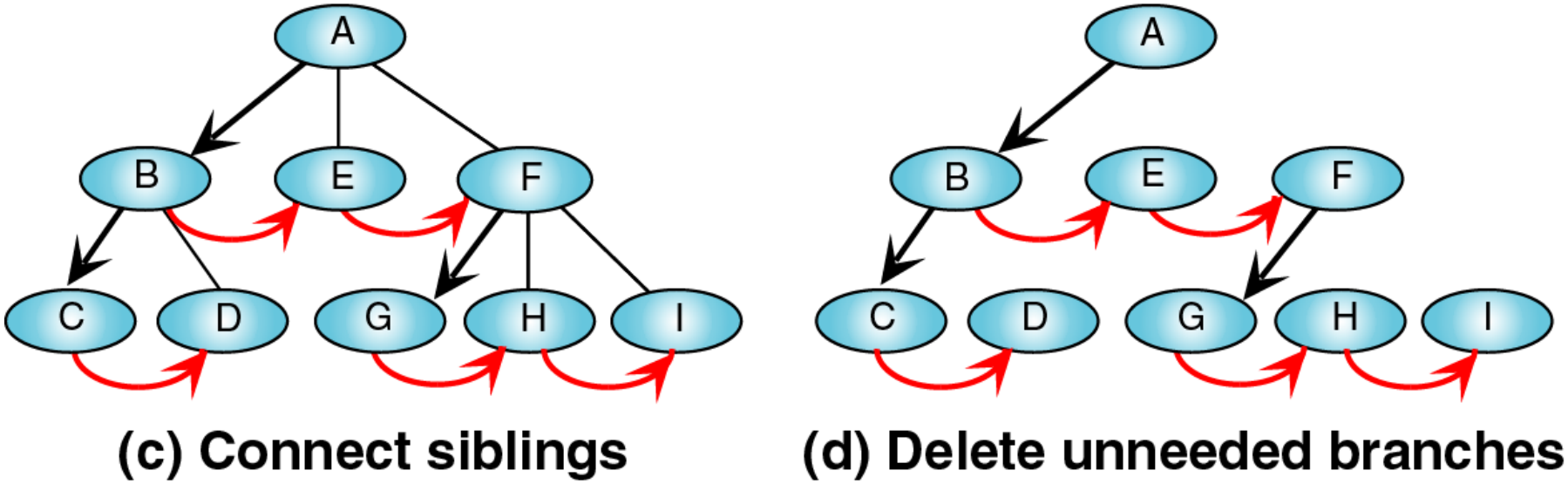


(a) The general tree

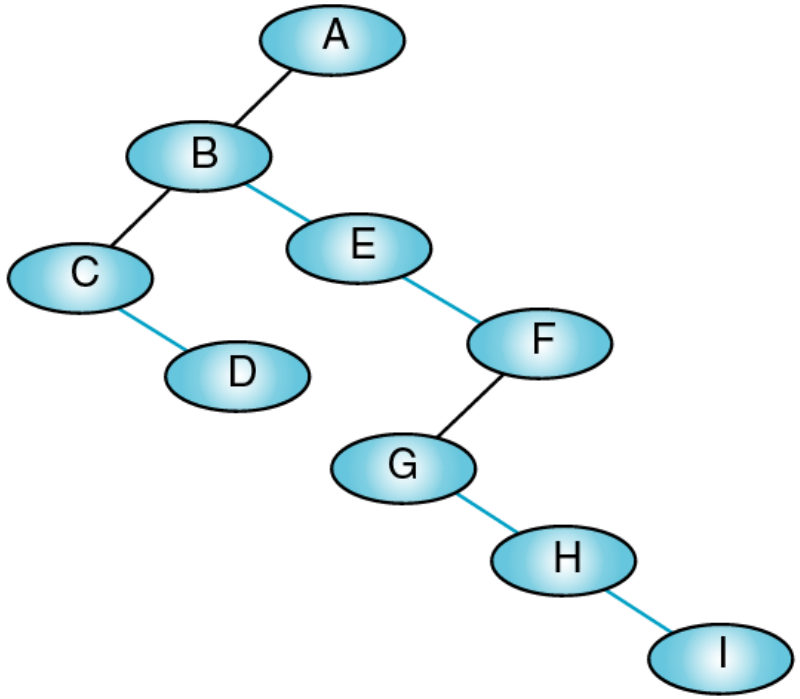


(b) Identify leftmost children

ตัวอย่างการแปลง General Tree เป็น Binary Tree



ตัวอย่างการแปลง General Tree เป็น Binary Tree



(e) The resulting binary tree

การสร้างทรีจาก Inorder และ Preorder

- จากหลักการทำงานของ Inorder ที่จะทำหน้าที่ในการแบ่งโหนดทางซ้ายขวาออกจากกัน
- ในขณะที่ Preorder จะทำหน้าที่ในการระบุตำแหน่งที่เป็น root ของแต่ละ subtree
- หากทราบ Inorder และ Preorder ของทรี จะทำให้สามารถสร้างภาพจำลองของทรีออกมาได้

การสร้างทรีจาก Inorder และ Preorder

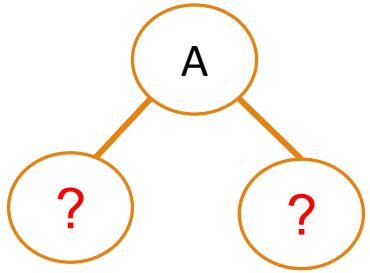
- Preorder : A B D C E F
- Inorder : D B A E C F

ขั้นตอนที่ 1 จาก Preorder จะได้ root ว่า คือ A

ขั้นตอนที่ 2 นำ A มาพิจารณาใน Inorder จะได้ว่า

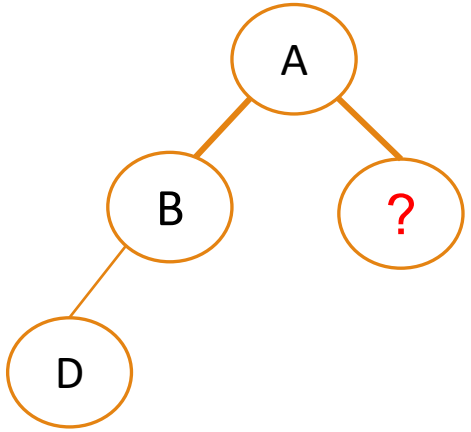
โหนดด้านซ้ายของ A คือ D B

โหนดด้านขวาของ A คือ E C F



การสร้างทรีจาก Inorder และ Preorder

- Preorder : A B D C E F
- Inorder : D B A E C F



ขั้นตอนที่ 3 พิจารณาโหนดทางด้านซ้ายของ A เพื่อหา root ของกลุ่ม

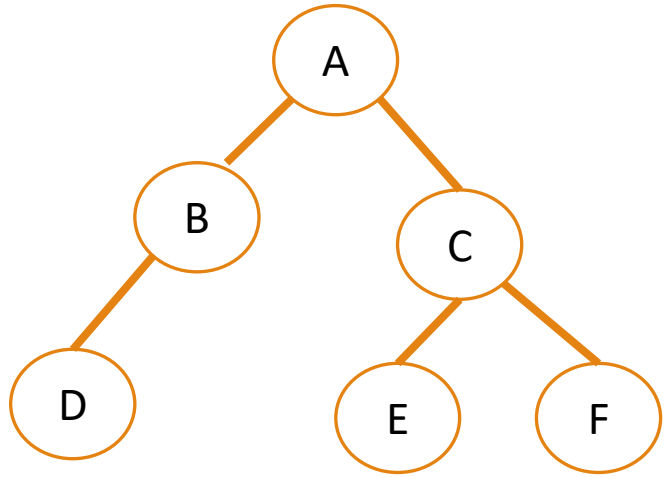
โดยกลับไปพิจารณาใน Preorder ว่าโหนดใดในกลุ่มมาก่อนกัน

จะพบว่า B มาก่อน B จึงเป็น root ของกลุ่ม

ขั้นตอนที่ 4 พิจารณา Inorder เพื่อหาตำแหน่งของ D ว่าเป็นโหนดซ้ายหรือขวาของ B พบว่า D อยู่ด้านซ้ายของ B จึงเป็นโหนดลูกด้านซ้าย

การสร้างทรีจาก Inorder และ Preorder

- Preorder : A B D C E F
- Inorder : D B A E C F



ขั้นตอนที่ 5 พิจารณาโหนดทางด้านขวาของ A เพื่อหา root ของกลุ่ม

โดยกลับไปพิจารณาใน Preorder ว่าโหนดใดในกลุ่มมาก่อนกัน

จะพบว่า C มาก่อน C จึงเป็น root ของกลุ่ม

ขั้นตอนที่ 4 พิจารณา Inorder เพื่อหาตำแหน่งของ E และ F ว่าเป็นโหนดซ้ายหรือขวาของ C พบว่า C อยู่ตรงกลางระหว่างโหนดทั้งสอง จึงได้ว่า E เป็นโหนดลูกด้านซ้าย และ F เป็นโหนดลูกด้านขวา