ต้นไม้แบบทวิภาค

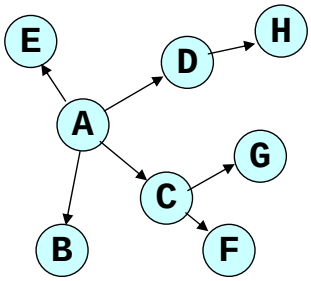
(Binary Trees)

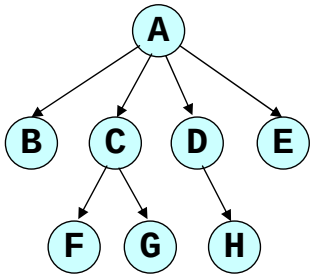
หัวข้อ

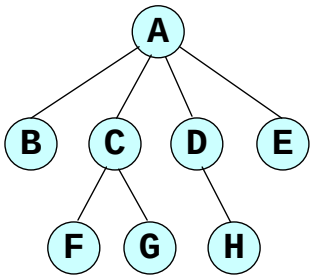
- นิยามต้นไม้
- การสร้างต้นไม้
- ต้นไม้แบบทวิภาค
  - ต้นไม้รหัสฮัฟฟ์แมน
  - ต้นไม้นิพจน์
  - การแวะผ่าน
  - การคำนวณค่าของต้นไม้นิพจน์
  - การหาอนุพันธ์ของฟังก์ชันตัวแปรเดียว

ต้นไม้ (Tree)

- ต้นไม้ประกอบด้วยปม (nodes) กับเส้นเชื่อม (edges)
- เส้นเชื่อมมีทิศทาง
- A เป็นปมพ่อของ B เมื่อมีเส้นเชื่อมจาก A ไปยัง B
- แต่ละปมมีปมพ่อได้เพียงปมเดียว (ยกเว้นปมพิเศษคือรากไม่มีพ่อ)
- ต้นไม้ที่มีปม v ปม ย่อมมี v – 1 เส้นเชื่อม

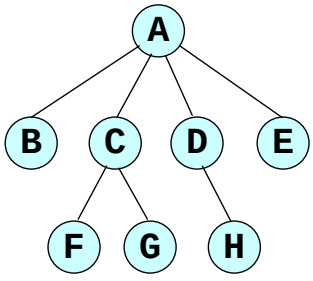


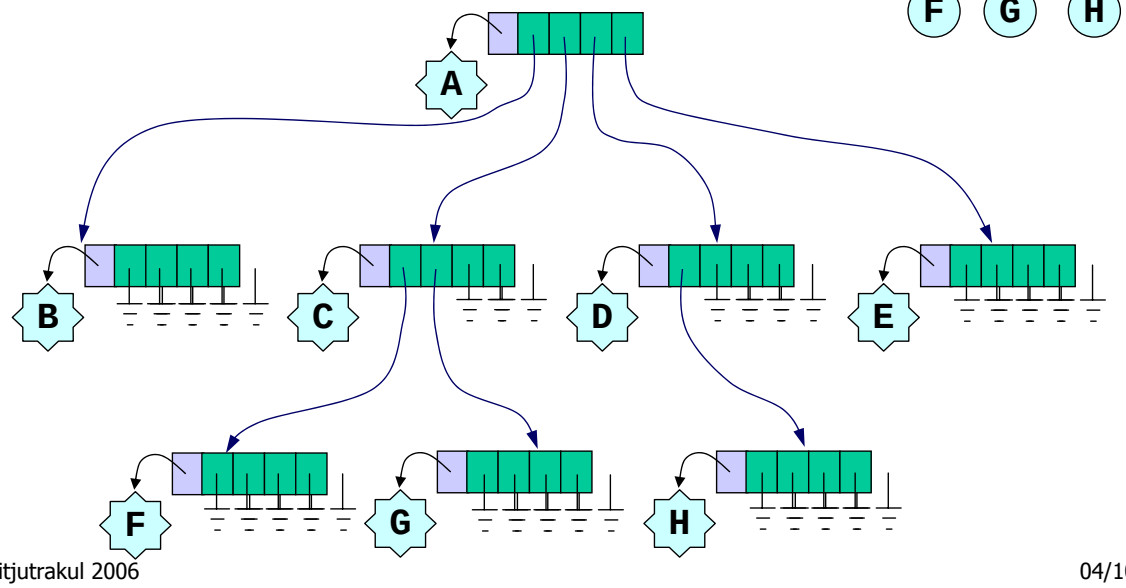




การสร้างต้นไม้ : ใช้อาเรย์เก็บลูก ๆ

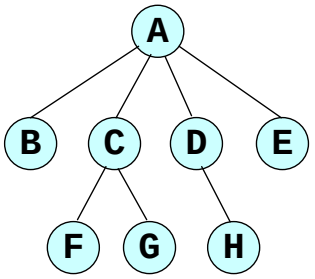
- ถ้ารู้จำนวนลูกมากสุดต่อปม
- ตัวเชื่อมส่วนมากเป็น null

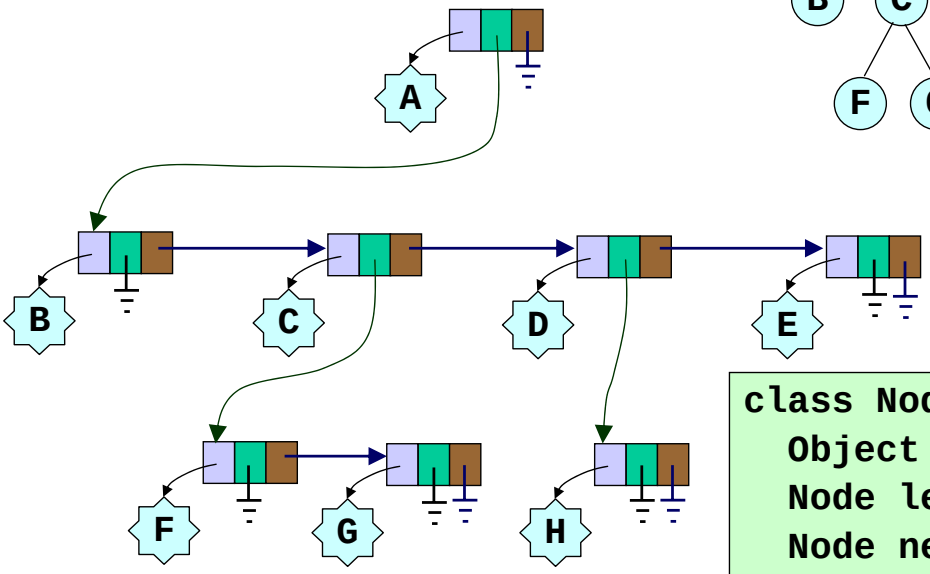




การสร้างต้นไม้ : ใช้รายการเก็บลูก ๆ

- มีตัวเชื่อมไปยังลูกคนโต
- มีตัวเชื่อมไปยังน้องคนถัดไป

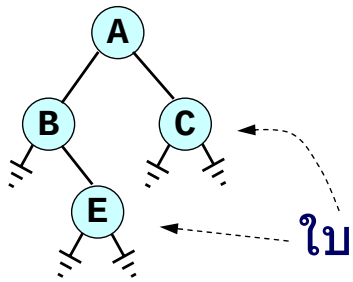


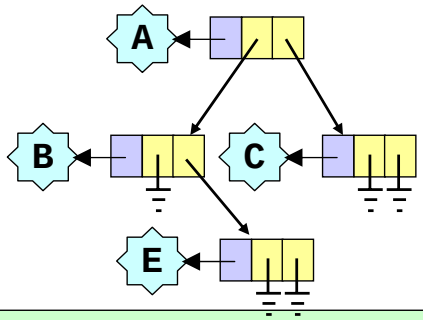



```
class Node {
  Object element;
  Node leftChild;
  Node nextSibling;
}
```


ต้นไม้แบบทวิภาค (Binary Tree)

- ทุกปมมีสองลูก : ลูกซ้าย และลูกขวา

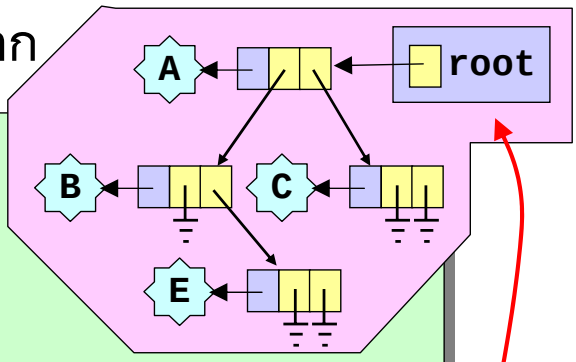





```
class Node {
  Object element;
  Node left, right;
  Node(Object e, Node l, Node r) {
    element = e; left = l; right = r;
  }
  boolean isLeaf() {
    return left == null && right == null;
  }
}
```

การสร้างต้นไม้แบบทวิภาค

- อ็อบเจกต์ต้นไม้เก็บเฉพาะราก




```
public class BinaryTree {
  static class Node {
    Object element;
    Node left;
    Node right;
    Node(Object e, Node l, Node r) {
      element = e; left = l; right = r;
    }
    boolean isLeaf() {
      return left == null && right == null;
    }
  }
  Node root;
  ...
```


รหัสฮัฟฟ์แมน (Huffman Code)

	'ช'	'ม'	'ย'	'ป'	'ส'	'า'
จำนวน	40	21	15	14	8	2
รหัสแบบความยาวคงที่	000	001	010	011	100	101
รหัสแบบความยาวแปรได้	0	100	101	110	1110	1111

10000100010101010001101

ส ม ช า ย ม า

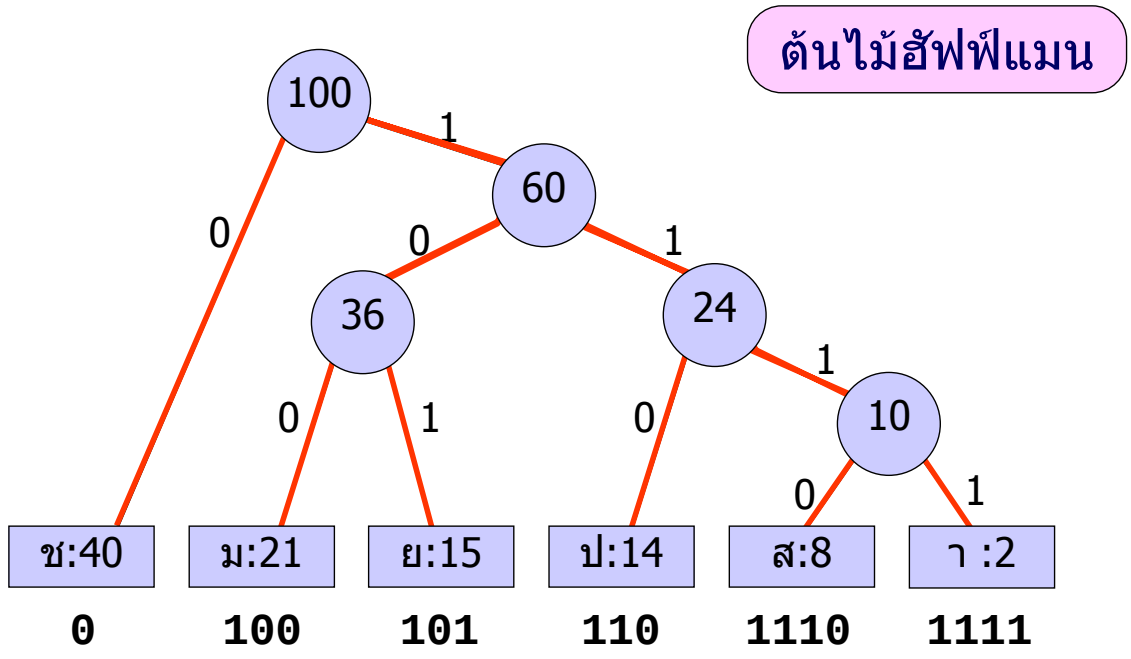
11101000111111011001111

ส ม ช า ย ม า

40×3 + 21×3 + 15×3 + 14×3 + 8×3 + 2×3 = 300

40×1 + 21×3 + 15×3 + 14×3 + 8×4 + 2×4 = 230

วิธีการหารหัสฮัฟฟ์แมน



รหัสฮัฟฟ์แมน

โปรแกรมการหาต้นไม้ฮัฟฟ์แมน

```
public static HuffmanTree coding(int[] freq) {
  BinaryMinHeap h = new BinaryMinHeap();
  for (int i = 0; i < freq.length; i++) {
    h.enqueue(new HuffmanTree(freq[i], null, null));
  }
  for (int i = 0; i < freq.length - 1; i++) {
    HuffmanTree t1 = (HuffmanTree) h.dequeue();
    HuffmanTree t2 = (HuffmanTree) h.dequeue();
    int f = t1.freq() + t2.freq();
    h.enqueue(new HuffmanTree(f, t1.root, t2.root));
  }
  return (HuffmanTree) h.dequeue();
}
```

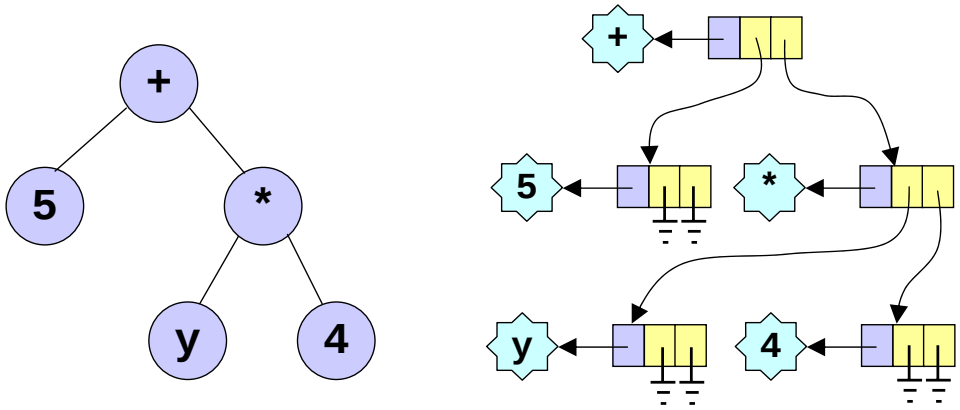
ต้นไม้ฮัฟฟ์แมน (HuffmanTree)

```
public class HuffmanTree extends BinaryTree
                         implements Comparable {
  ...
  public HuffmanTree(int freq, Node left, Node right) {
    root = new Node(new Integer(freq), left, right);
  }
  public int freq() {
    return ((Integer) root.element).intValue();
  }
  public int compareTo(Object obj) {
    return freq() - ((HuffmanTree) obj).freq();
  }
}
```


ต้นไม้นิพจน์ (Expression Tree)

- แทนนิพจน์ได้ด้วยต้นไม้
- ใบ : ตัวถูกดำเนินการ (operand)
- ปม : ตัวดำเนินการ (operator)

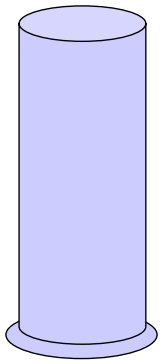
5 + y * 4



การสร้างต้นไม้นิพจน์

- พบ operand ให้ push ลงกองซ้อน
- พบ operator ให้ pop ออกมาเป็นลูกของปมใหม่ เพื่อ push ลงกองซ้อน

```
infix : 5  +  y  *  4
postfix : 5  y  4  *  +
```



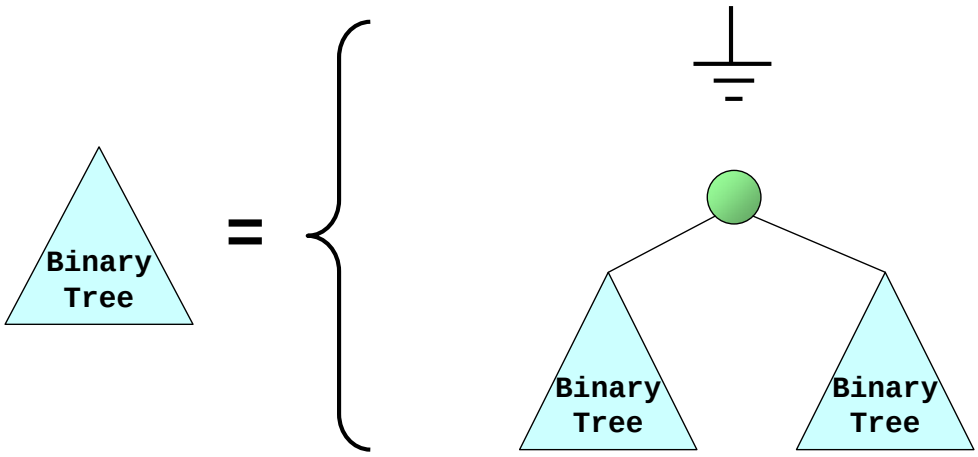
โปรแกรมการสร้างต้นไม้นิพจน์

```
public class Expression extends BinaryTree {
  public Expression(List infix) {
    List postfix = infix2Postfix(infix);
    Stack s = new ArrayStack(10);
    for (int i=0; i<postfix.size(); i++) {
      String token = (String)postfix.get(i);
      if (!isOperator(token)) {
        s.push(new Node(token, null, null));
      } else {
        Node right = (Node) s.pop();
        Node left = (Node) s.pop();
        s.push(new Node(token, left, right));
      }
    }
    root = (Node) s.pop();
  }
  ...
```

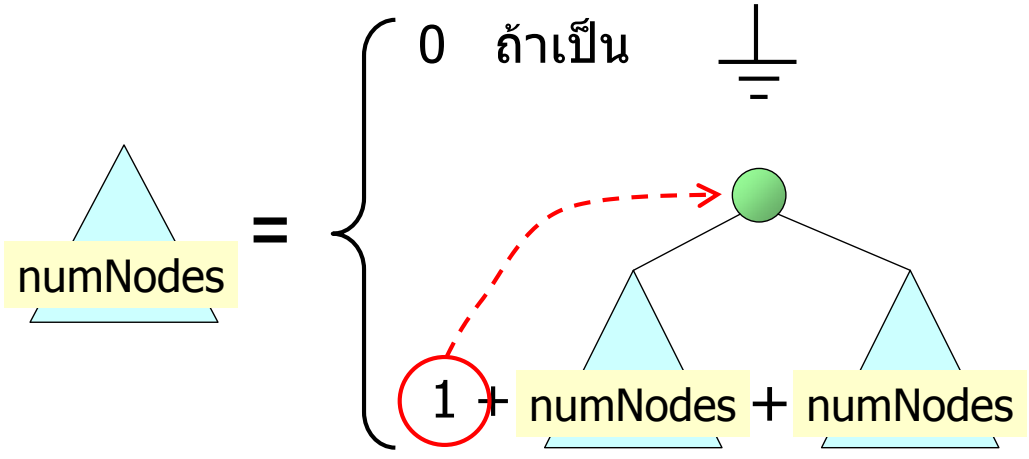
pop 2 ตัวเพราะเป็น binary operaor

มองต้นไม้แบบทวิภาคแบบเวียนเกิด

- Binary tree คือ
  - ต้นไม้ว่าง (null) หรือ
  - หนึ่งปม และลูกต้นซ้ายกับลูกต้นขวา

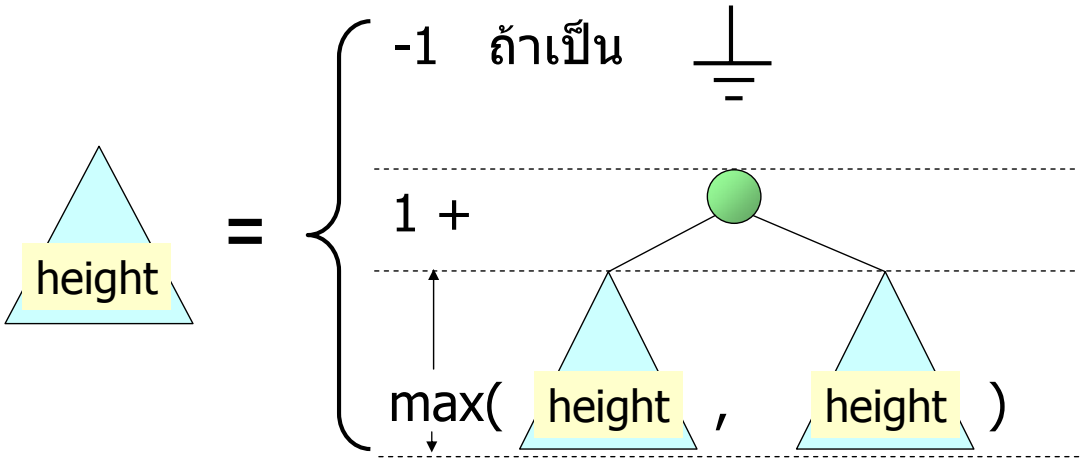


การหาจำนวนปมทั้งหมดของต้นไม้




```
int numNodes(Node node) {
  if ( node == null ) return 0;
  return 1 + numNodes(node.left) +
             numNodes(node.right);
}
```

การหาความสูงของต้นไม้

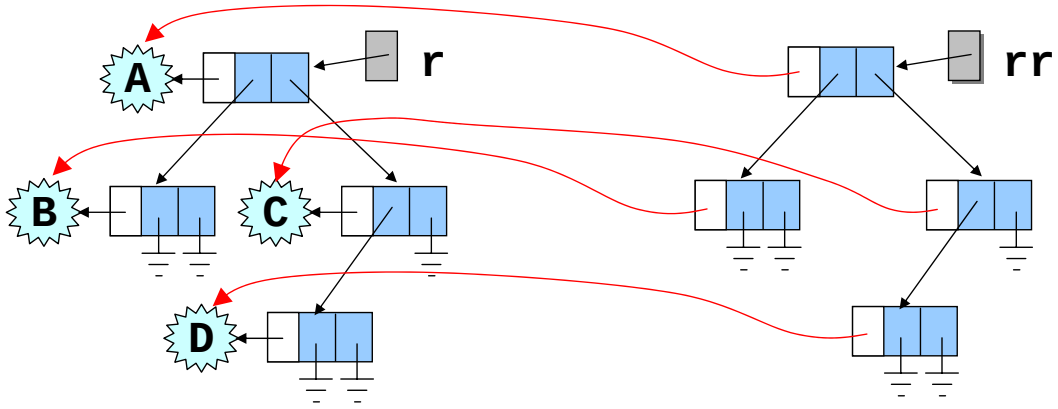



```
int height(Node node) {
  if ( node == null ) return -1;
  return 1 + Math.max(height(node.left),
                      height(node.right));
}
```


คลาส BinaryTree

```
public class BinaryTree {
  static class Node { ... }
  Node root;
  public int numNodes() {
    return numNodes(root);
  }
  public int height() {
    return height(root);
  }
  private int numNodes(Node node) {
    if ( node == null ) return 0;
    return 1 + numNodes(node.left) +
               numNodes(node.right);
  }
  private int height(Node node) {
    if ( node == null ) return -1;
    return 1 + Math.max(height(node.left),
                        height(node.right));
  }
```

การสำเนาต้นไม้




```
Node rr = copy(r);
```

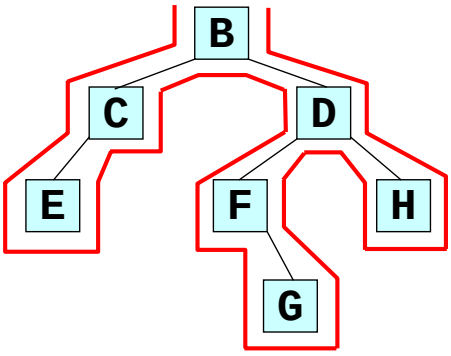

การสำเนาต้นไม้ (มองแบบเวียนเกิด)

```
public class Expression extends BinaryTree {
  ...
  // copy constructor
  public Expression(Expression e) {
    root = copy(e.root);
  }
  ...
}
```

```
Node copy(Node r) {
  if (r == null) return null;
  Node leftTree = copy(r.left);
  Node rightTree = copy(r.right);
  return new Node(r.element, leftTree, rightTree);
}
```

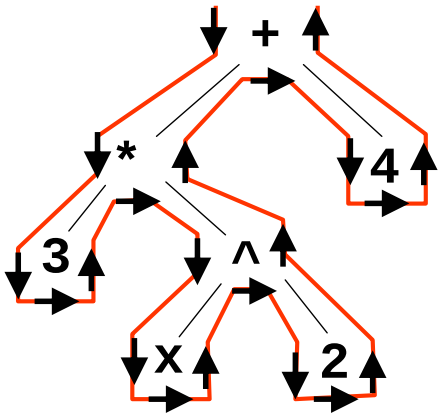
การแวะผ่านต้นไม้

- Tree traversal เป็นกระบวนการเข้าถึงปมต่าง ๆ ในต้นไม้ ปมละหนึ่งครั้งอย่างมีระเบียบ
  - แบบก่อนลำดับ (preorder) B, C, E, D, F, G, H
  - แบบตามลำดับ (inorder) E, C, B, F, G, D, H
  - แบบหลังลำดับ (postorder) E, C, G, F, H, D, B



การแวะผ่านต้นไม้นิพจน์

- แวะผ่านแบบก่อนลำดับ ได้นิพจน์แบบ prefix
- แวะผ่านแบบตามลำดับ ได้นิพจน์แบบ infix
- แวะผ่านแบบหลังลำดับ ได้นิพจน์แบบ postfix



+ * 3 ^ x 2 4

3 * x ^ 2 + 4

3 x 2 ^ * 4 +

การแวะผ่านต้นไม้ที่มี x เป็นราก

- แบบก่อนลำดับ
  - แวะ x
  - แวะผ่าน x.left
  - แวะผ่าน x.right
- แบบตามลำดับ
  - แวะผ่าน x.left
  - แวะ x
  - แวะผ่าน x.right
- แบบหลังลำดับ
  - แวะผ่าน x.left
  - แวะผ่าน x.right
  - แวะ x

```
void preOrder(Node x) {
  if (x == null) return;
  visit(x.element);
  preorder(x.left);
  preorder(x.right);
}
```

```
void inOrder(Node x) {
  if (x == null) return;
  inorder(x.left);
  visit(x.element);
  inorder(x.right);
}
```

```
void postOrder(Node x) {
  if (x == null) return;
  postorder(x.left);
  postorder(x.right);
  visit(x.element);
}
```


toArray( )

- คืนอาเรย์ที่บรรจุข้อมูลตามปมต่าง ๆ ทุกปมในต้นไม้

```
public class BinaryTree {
  Node root;
  ...
  public Object[] toArray() {
    int n = numNodes(root);
    Object[] a = new Object[n];
    toArray(root, a, 0);
    return a;
  }
  private int toArray(Node x, Object[] a, int k) {
    if (x == null) return k;
    a[k++] = x.element;
    k = toArray(x.left, a, k);
    k = toArray(x.right, a, k);
    return k;
  }
```

นำข้อมูลตามปมในต้นไม้ root ไปใส่ในอาเรย์ a เริ่มตั้งแต่ช่อง 0 เป็นต้นไป

มีโครงการทำงานแบบ preorder

เขียน toArray แบบใช้ Visitor

```
public abstract class Visitor {
  public abstract void visit(Object e);
}
```

```
void preOrder(Node r, Visitor v) {
  if (r == null) return;
  v.visit(r.element);
  preOrder(r.left, v);
  preOrder(r.right, v);
}
```

```
public Object[] toArray() {
  final Object[] a = new Object[numNodes()];
  Visitor v = new Visitor() {
                int k = 0;
                public void visit(Object e) {
                  a[k++] = e;
                }
              };
  preOrder(root, v);
  return a;
}
```

anonymous class

สร้างอ็อบเจกต์ของคลาส
ที่เป็นคลาสลูกของ Visitor

ปรับปรุงให้ Visitor ยุติการแวะผ่านได้

```
public abstract class Visitor {
  private boolean done = false;
  public void done() {
    done = true;
  }
  public boolean isDone() {
    return done;
  }
  public abstract void visit(Object e);
}
```

```
void preOrder(Node r, Visitor v) {
  if (r == null || v.isDone()) return;
  v.visit(r.element);
  preOrder(r.left, v);
  preOrder(r.right, v);
}
```

การตรวจสอบว่าต้นไม้เก็บ x หรือไม่

```
public boolean contains(final Object x) {
  Visitor v = new Visitor() {
                public void visit(Object e) {
                  if (e.equals(x)) done();
                }
              };
  preOrder(root, v);
  return v.isDone();
}
```

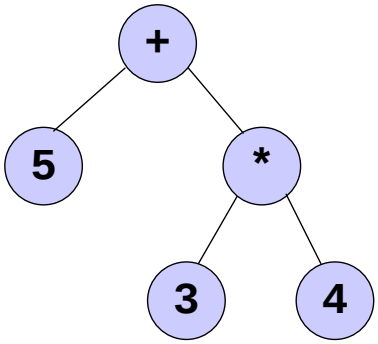

การแวะผ่านด้วย Visitor

```
public class BinaryTree {
  ...
  public void preOrder(Visitor v)  { preOrder(root, v);  }
  public void inOrder(Visitor v)   { inOrder(root, v);   }
  public void postOrder(Visitor v) { postOrder(root, v); }

  void preOrder(Node x, Visitor v) {
    if (x == null || v.isDone()) return;
    v.visit(x.element);
    preOrder(x.left, v);
    preOrder(x.right, v);
  }
  void inOrder(Node x, Visitor v) {
    if (x == null || v.isDone()) return;
    inOrder(x.left, v);
    v.visit(x.element);
    inOrder(x.right, v);
  }
  void postOrder(Node x, Visitor v) {
    ...
```

การคำนวณค่าของต้นไม้นิพจน์

- ต้องรู้ค่าของลูก ๆ ก่อน จึงคำนวณได้
- มีลักษณะคล้ายการแวะผ่านแบบหลังลำดับ



เมท็อดการคำนวณค่าของต้นไม้นิพจน์

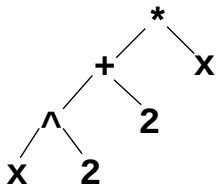
```
public class Expression extends BinaryTree {
  ...
  public double eval() {
    return eval(root);
  }
  private double eval(Node r) {
    if (r == null) return 0;
    if (r.isLeaf())
      return Double.parseDouble((String) r.element);
    double vLeft = eval(r.left);
    double vRight = eval(r.right);
    if (r.element.equals("+")) return vLeft + vRight;
    if (r.element.equals("-")) return vLeft – vRight;
    if (r.element.equals("*")) return vLeft * vRight;
    if (r.element.equals("/")) return vLeft / vRight;
    if (r.element.equals("^")) return Math.pow(vLeft,vRight);
    throw new IllegalStateException();
  }
```

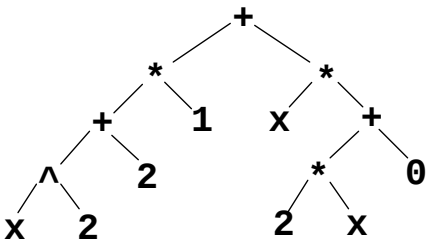
การหาอนุพันธ์ฟังก์ชันตัวแปรเดียว

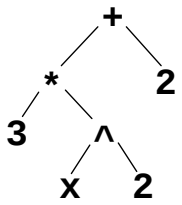
$$f(x) = (x^2 + 2)x$$

$$f'(x) = (x^2 + 2)\cdot 1 + x\cdot(2x + 0)$$

$$= 3x^2 + 2$$







f **f.diff()** **f.simplify()**

สูตรต่าง ๆ

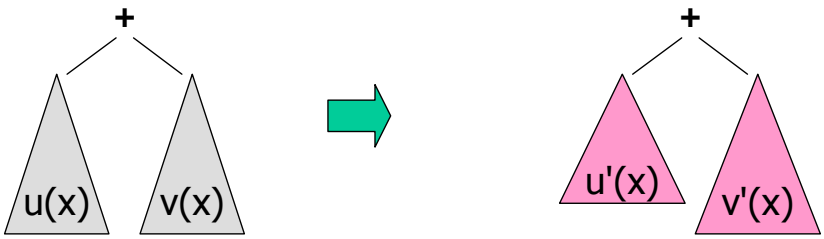
$$\big(u(x)+v(x)\big)' = u'(x)+v'(x)$$

$$\big(u(x)v(x)\big)' = v(x)u'(x)+u(x)v'(x)$$

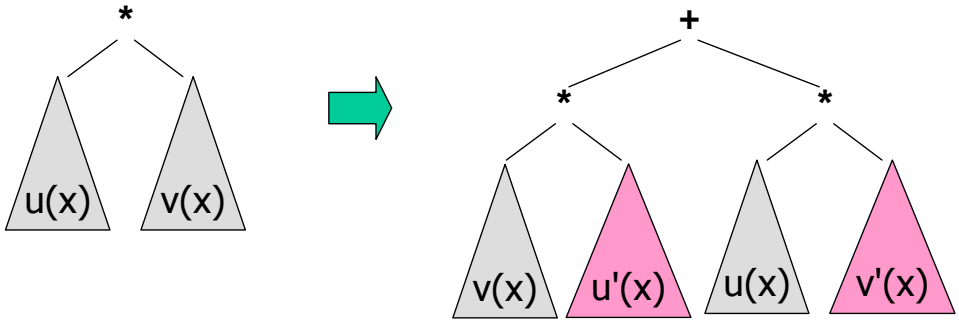
$$\left(\frac{u(x)}{v(x)}\right)' = \frac{v(x)u'(x)-u(x)v'(x)}{\big(v(x)\big)^2}$$

$$\Big(\big(u(x)\big)^C\Big)' = C\big(u(x)\big)^{C-1}u'(x)$$

อนุพันธ์ของต้นไม้นิพจน์ : +, *

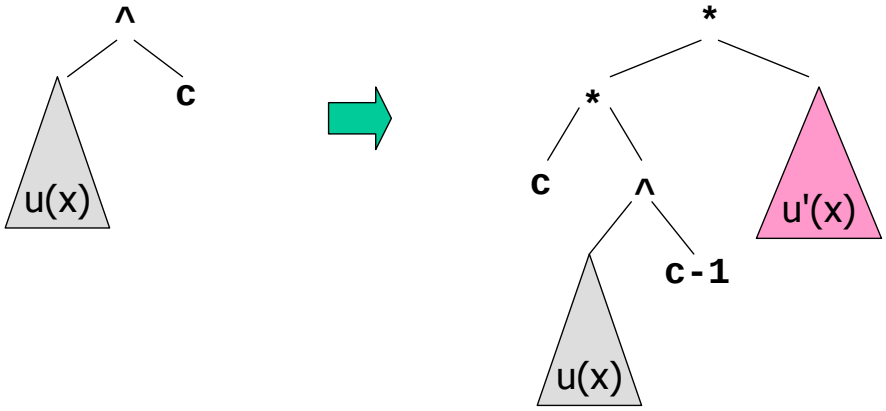


$$\left(u(x)+v(x)\right)' = u'(x)+v'(x)$$



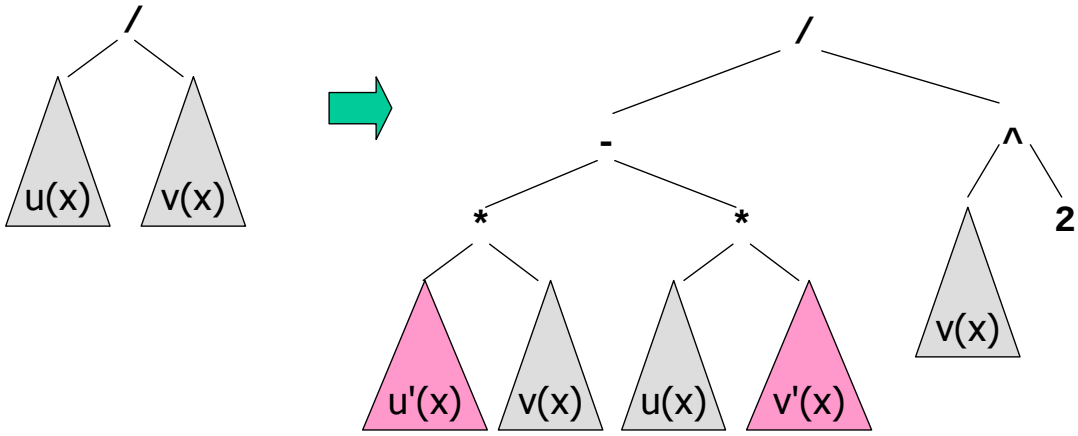
$$\left(u(x)v(x)\right)' = v(x)u'(x)+u(x)v'(x)$$

อนุพันธ์ของต้นไม้นิพจน์ : ^



$$\left(\left(u(x)\right)^C\right)' = C\left(u(x)\right)^{C-1}u'(x)$$

อนุพันธ์ของต้นไม้นิพจน์ : /



$$\left(\frac{u(x)}{v(x)}\right)' = \frac{v(x)u'(x) - u(x)v'(x)}{\left(v(x)\right)^2}$$

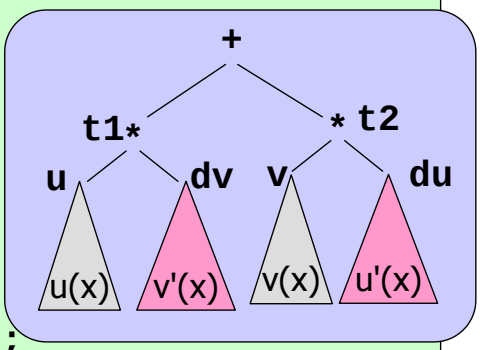
เมท็อดการหาอนุพันธ์

```
public class Expression extends BinaryTree {
  ...
  public void diff() {
    root = diff(root);
  }
  private Node diff(Node r) {
    if (r == null) return null;
    String s = (String) r.element;
    if ( r.isLeaf() ) {
      r.element = (s.equals("x") ? "1" : "0");
    } else {
      if (s.equals("+"))      r = diffSum(r);
      else if (s.equals("-")) r = diffSum(r);
      else if (s.equals("^")) r = diffExpo(r);
      else if (s.equals("*")) r = diffMult(r);
      else if (s.equals("/")) r = diffDiv(r);
    }
    return r;
  }
```

เมท็อดการหาอนุพันธ์ : +, *

```
public class Expression extends BinaryTree {
  ...
  private Node diffSum(Node r) {
    r.left = diff(r.left);
    r.right = diff(r.right);
    return r;
  }
  private Node diffMult(Node r) {
    Node u = copy(r.left);
    Node v = copy(r.right);
    Node du = diff(r.left);
    Node dv = diff(r.right);
    Node t1 = new Node("*", u, dv);
    Node t2 = new Node("*", v, du);
    return new Node("+", t1, t2);
  }
```

$$\left(u(x)+v(x)\right)' = u'(x)+v'(x)$$



$$\left(u(x)v(x)\right)' = v(x)u'(x)+u(x)v'(x)$$

สรุป

- ต้นไม้เป็นโครงสร้างในการจัดเก็บข้อมูล
  - สร้างต้นไม้ได้ด้วยการโยงปมของต้นไม้
- ต้นไม้แบบทวิภาคเป็นต้นไม้ที่แต่ละปมมี 2 ลูก
  - มองต้นไม้ใหญ่ประกอบด้วยต้นไม้ย่อย
  - ทำให้เขียนเมท็อดได้แบบเวียนเกิด
  - การประมวลข้อมูลตามปม กระทำได้ด้วยการแวะผ่านแบบก่อนลำดับ ตามลำดับ หรือหลังลำดับ