2110211 โครงสร้างข้อมูลเบื้องต้น

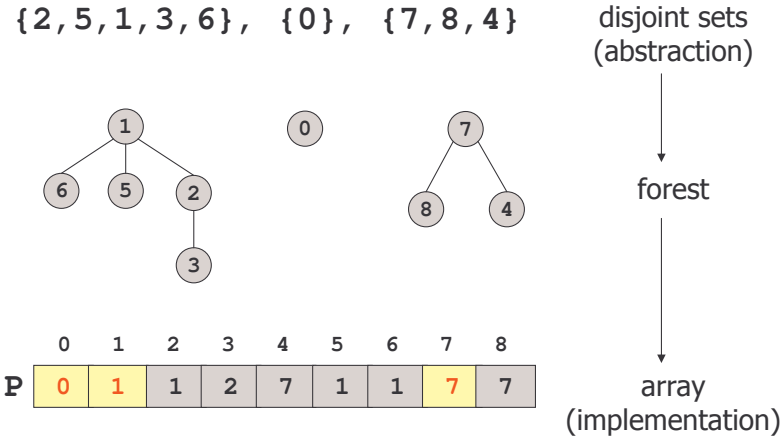
Binary Tree

สมชาย ประสิทธิ์จูตระกูล

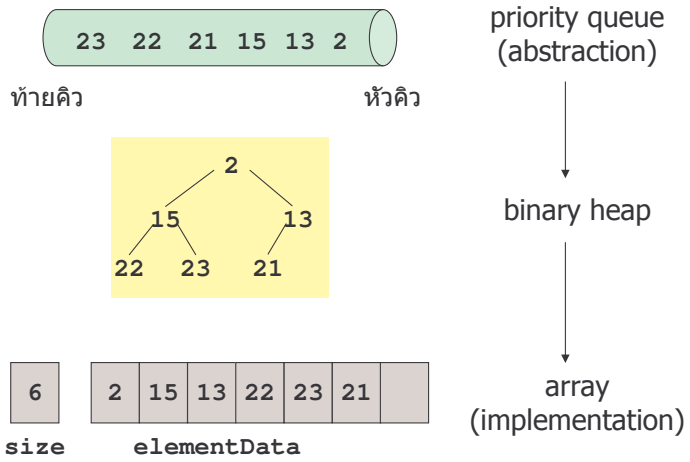
ต้นไม้

- ต้นไม้คือกราฟที่ไม่มีวงจร
- rooted tree
  - nodes ต่าง ๆ มีความสัมพันธ์พ่อ-ลูก
  - แต่ละ node มี 1 parent node ยกเว้น root node
- ordered tree
  - rooted tree ที่ลูก ๆ มีอันดับ (ลูกคนที่ 1, ลูกคนที่ 2, ...)

ตัวอย่างการใช้ต้นไม้ : Disjoint Sets

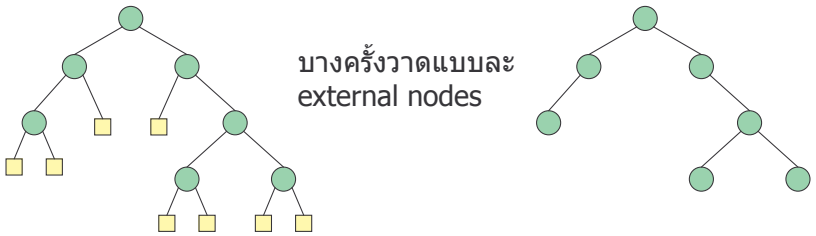


ตัวอย่างการใช้ต้นไม้ : Priority Queue



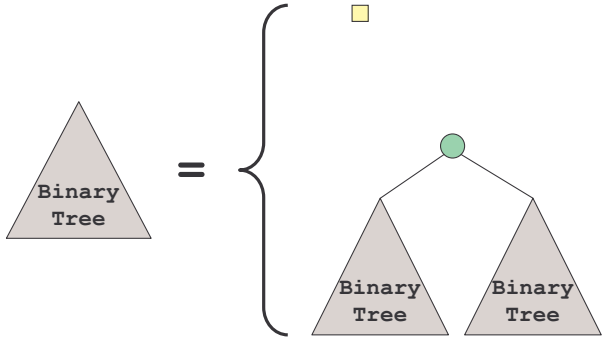
Binary Tree

- ordered tree
- มี nodes สองประเภท
  - external nodes : node ที่ไม่มีลูกเลย
  - internal nodes : node ที่มีสองลูก คือ ลูกซ้ายและลูกขวา



Binary Tree (นิยามแบบ recursive)

- Binary tree คือ
  - 1 external node หรือ
  - 1 internal node ที่ลูกซ้ายและลูกขวาเป็นต้นไม้ย่อยแบบ binary tree ทั้งคู่



คุณสมบัติของ Binary Trees

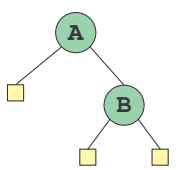
- ให้ T_X คือ binary tree ที่รากคือ node x
- L(x) คือลูกซ้ายของ node x
- R(x) คือลูกขวาของ node x

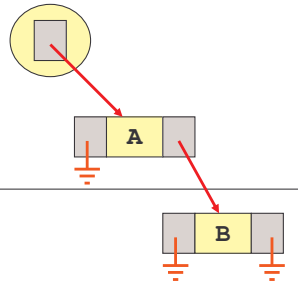
$$\#nodes(T_x) = \begin{cases} 0 & \text{if x is an external node} \\ 1 + \#nodes(T_{L(x)}) + \#nodes(T_{R(x)}) & \end{cases}$$

internal nodes

$$height(T_x) = \begin{cases} -1 & \text{if x is an external node} \\ 1 + \max(height(T_{L(x)}), height(T_{R(x)})) & \end{cases}$$

วิธีสร้าง binary tree






```
class BinaryNode {
  Object element;
  BinaryNode left;
  BinaryNode right;

  BinaryNode(Object e, BinaryNode l, BinaryNode r) {
    this.element = e;
    this.left = l;
    this.right = r;
  }
}
```

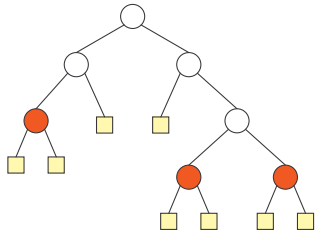
```
public class BinaryTree {
  private BinaryNode root;

  ...
}
```

Binary Trees : height(), size()

```
public class BinaryTree {
  private BinaryNode root;

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

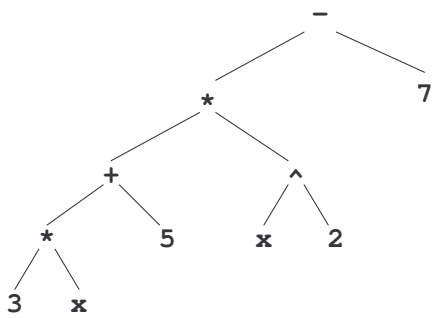

การบ้าน : numLeaves()

- เขียน numLeaves( ) ใน BinaryTree ซึ่งคืนจำนวนใบในต้นไม้
- ใบคือ internal node ที่มีลูกทั้งสองเป็น external nodes (เช่น ต้นไม้ข้างล่างนี้มี 3 ใบ)



ตัวอย่างการใช้งาน Binary Tree

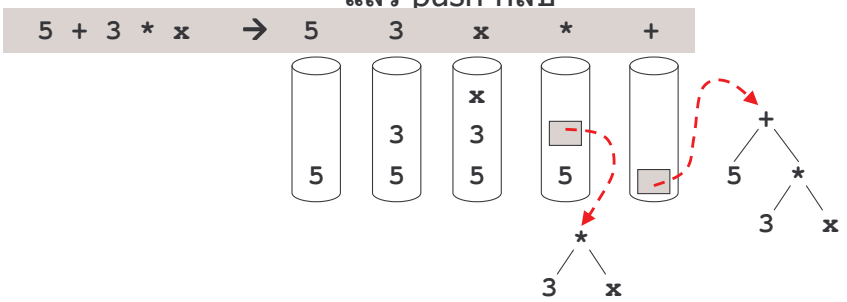
```
(3 * x + 5) * x ^ 2 - 7
```



Expression Tree

วิธีสร้าง Expression Tree

- เปลี่ยน infix เป็น postfix ก่อน
- สร้าง expression tree จาก postfix (ทำเหมือน postfix evaluation)
  - ดูแต่ละตัวใน postfix จากซ้ายไปขวา
  - พบ operand : ให้ push ลง stack
  - พบ operator : ให้ pop operands มาสร้างต้นไม้ย่อย แล้ว push กลับ



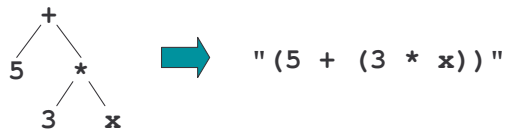
วิธีสร้าง Expression Tree

```
public class ExpressionTree {
  static enum OP {ADD, SUB, MUL, DIV, POW};
  private BinaryNode root;

  public ExpressionTree(List infix) {
    List postfix = infixToPostfix(infix);
    Stack s = new ArrayStack();
    for (int i = 0; i < postfix.size(); i++) {
      Object token = postfix.get(i);
      if (!(token instanceof OP)) {
        s.push(new BinaryNode(token, null, null));
      } else {
        BinaryNode right = (BinaryNode) s.pop();
        BinaryNode left = (BinaryNode) s.pop();
        s.push(new BinaryNode(token, left, right));
      }
    }
    root = (BinaryNode) s.pop();
  }
```


ExpressionTree : toString( )

```
  public String toString() {
    return toString(root);
  }
  private static String toString(BinaryNode r) {
    if (r == null) return "";
    if (r.left == null && r.right == null)
      return r.element.toString();
    else
      return "(" + toString(r.left) +
             " " + r.element.toString() +
             " " + toString(r.right) + ")";
  }
```

ต้องแก้ไข enum OP ให้มี toString ด้วยสำหรับค่าต่าง ๆ ใน enum




```
static enum OP {
  ADD {
    double eval(double d1, double d2) {return d1 + d2;}
    public String toString() {return "+";}
  },
  SUB {
    double eval(double d1, double d2) {return d1 - d2;}
    public String toString() {return "-";}
  },
  MUL {
    double eval(double d1, double d2) {return d1 * d2;}
    public String toString() {return "*";}
  },
  DIV {
    double eval(double d1, double d2) {return d1 / d2;}
    public String toString() {return "/";}
  },
  POW {
    double eval(double d1, double d2) {return Math.pow(d1,d2);}
    public String toString() {return "^";}
  };
  abstract double eval(double d1, double d2);
}
```

ปรับจาก enum OP ที่เคยเขียนในเรื่อง stack

ExpressionTree : copy constructor

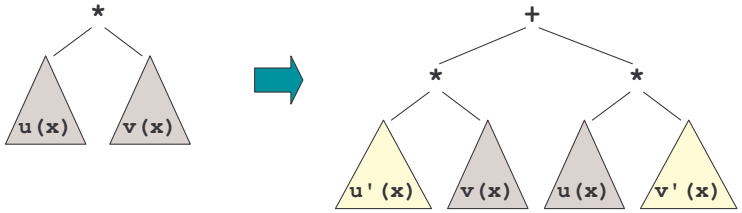
```
public class ExpressionTree {
  private BinaryNode root;

  public ExpressionTree(ExpressionTree t) {
    if (t.root != null) root = new BinaryNode(t.root);
  }
  ...
```

```
class BinaryNode {
  Object element;
  BinaryNode left, right;
  ...
  BinaryNode(BinaryNode n) {
    element = n.element;
    if (n.left != null) left = new BinaryNode(n.left);
    if (n.right != null) right = new BinaryNode(n.right);
  }
  ...
}
```

รับต้นไม้หนึ่งต้น แล้ว deep-clone อีกต้นคืนให้

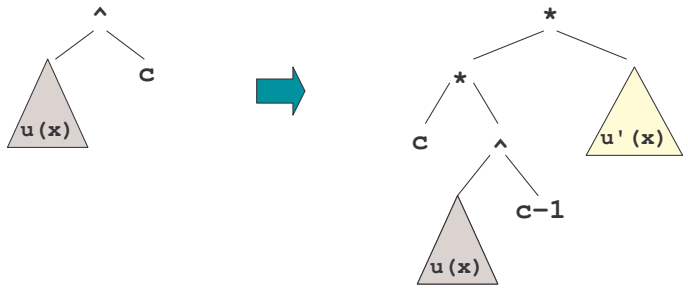
หาอนุพันธ์

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



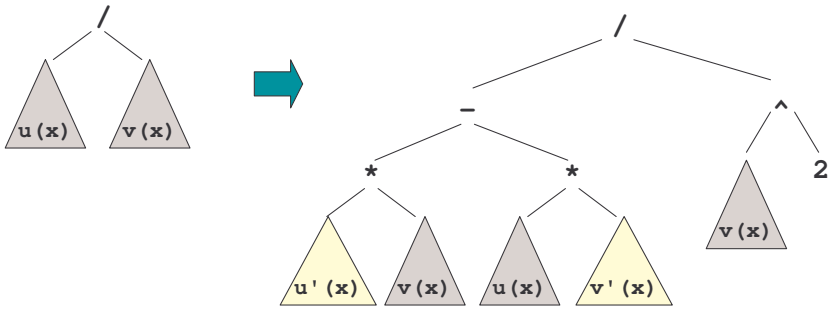
$$\left(u(x)v(x)\right)' = v(x)u'(x)+u(x)v'(x)$$

หาอนุพันธ์



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

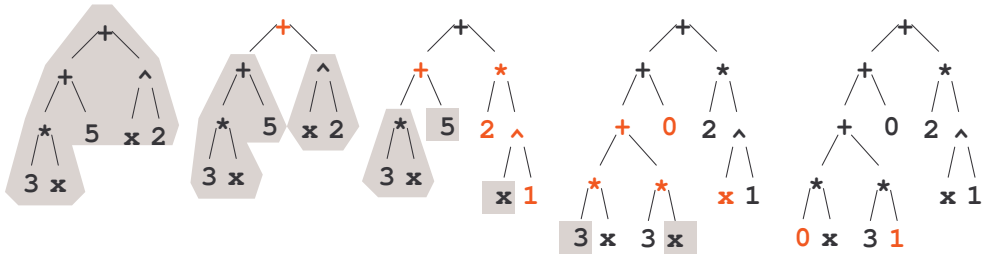
หาอนุพันธ์



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

หาอนุพันธ์

- t = expression tree ของ 3 * x + 5 + x ^ 2
- t.diff( )
  - ทำให้ t เป็น expression tree ของ 3 + 2 * x

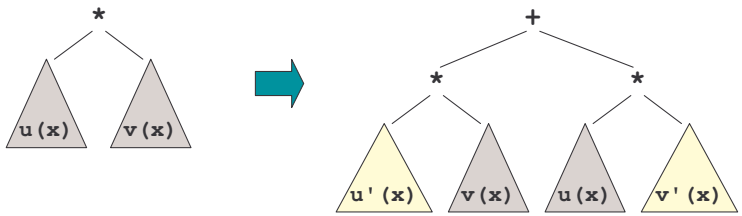


```
public class ExpressionTree {
  private BinaryNode root;
  ...
  public void diff() {
    root = diff(root);
  }
  private static BinaryNode diff(BinaryNode r) {
    if (r == null) return null;
    Object e = r.element;
    if (e instanceof OP) {
      if (e == OP.ADD || e == OP.SUB) r = diffAdd(r);
      else if (e == OP.POW) r = diffPow(r);
      else if (e == OP.MUL) r = diffMul(r);
      else if (e == OP.DIV) r = diffDiv(r);
      else assert false;
    } else if (n.element instanceof Double) {
      r.element = new Double(0);
    } else {
      r.element = new Double(1);
    }
    return r;
  }
  ...
```



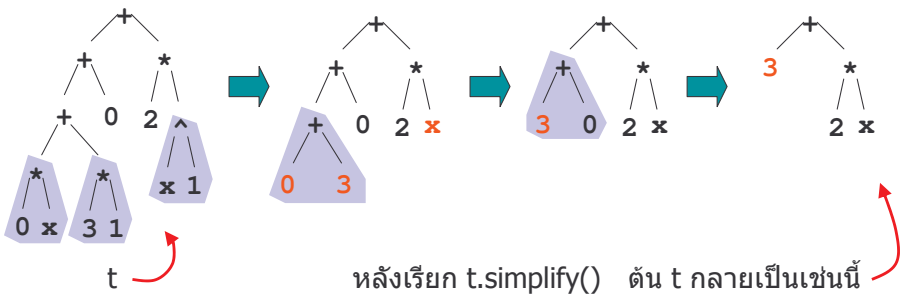
```
  private static BinaryNode diffAdd(BinaryNode r) {
    r.left = diff(r.left);
    r.right = diff(r.right);
    return r;
  }
  private static BinaryNode diffMul(BinaryNode r) {
    BinaryNode u = new BinaryNode(r.left);
    BinaryNode v = new BinaryNode(r.right);
    BinaryNode du = diff(r.left);
    BinaryNode dv = diff(r.right);
    BinaryNode vdu = new BinaryNode(OP.MUL, v, du);
    BinaryNode udv = new BinaryNode(OP.MUL, u, dv);
    return new BinaryNode(OP.ADD, vdu, udv);
  }
```

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



การบ้าน : ExpressionTree

- เขียน diffPow และ diffDiv ที่เหลือ
- ต้นไม้ที่ได้หลังเรียก diff ไม่สวย เขียนเมท็อด simpify( ) ที่เปลี่ยนต้นไม้ให้เล็กลง เช่น




```
public static void main(String[] a) {
   List infix = new ArrayList();
   String var = "x";
   infix.add(OP.OPENPAREN);
   infix.add(new Double(4));   infix.add(OP.MUL);
   infix.add(var);             infix.add(OP.POW);
   infix.add(new Double(2));   infix.add(OP.SUB);
   infix.add(new Double(3));   infix.add(OP.CLOSEPAREN);
   infix.add(OP.DIV);          infix.add(var);
   ExpressionTree t = new ExpressionTree(infix);
   System.out.println("f(x)  = " + t);
   t.diff();
   System.out.println("f'(x) = " + t);
   t.simplify();
   System.out.println("f'(x) = " + t);
 }
```

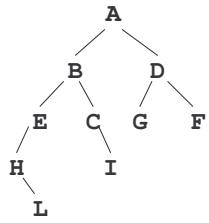
```
f(x)  = (((4.0 * (x ^ 2.0)) - 3.0) / x)
f'(x) = (((x * ((((x ^ 2.0) * 0.0) + (4.0 * ((2.0 * (x ^ (2.0 - 1.0)))
* 1.0))) - 0.0)) - (((4.0 * (x ^ 2.0)) - 3.0) * 1.0)) / (x ^ 2.0))
f'(x) = (((x * (8.0 * x)) - ((4.0 * (x ^ 2.0)) - 3.0)) / (x ^ 2.0))
```

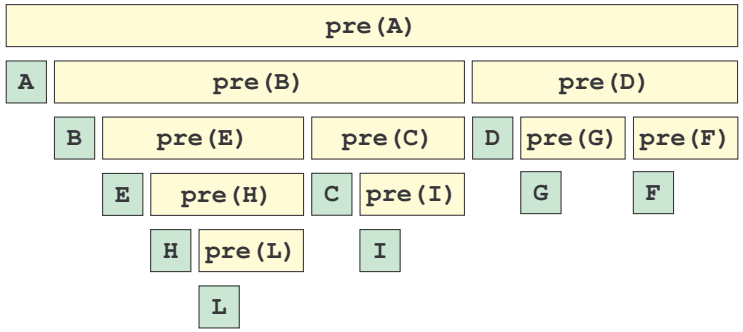
Tree Traversals

- tree traversal คือวิธีวิ่งไปตาม nodes ต่าง ๆ ในต้นไม้อย่างมีระบบให้ครบทุก node
- สามารถประมวลผลข้อมูลที่เก็บในต้นไม้ระหว่างการ traverse
- binary tree traversals แบบมาตรฐาน
  - preorder
  - inorder
  - postorder

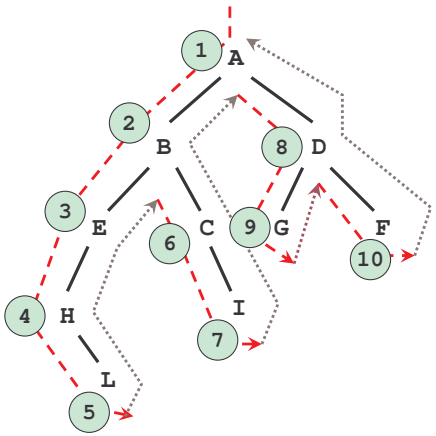
Preorder Traversal

$$pre(T_x) = \begin{cases} "" & \text{if x is an external node} \\ x + pre(T_{L(x)}) + pre(T_{R(x)}) & \end{cases}$$





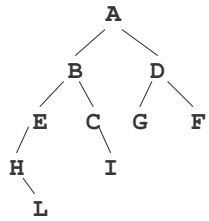
Preorder Traversal

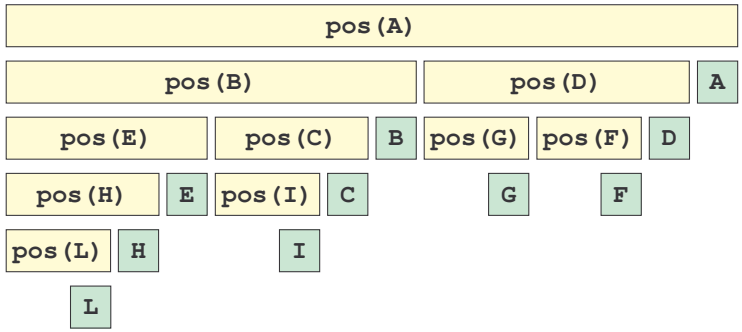


preorder : A, B, E, H, L, C, I, D, G, F

Postorder Traversal

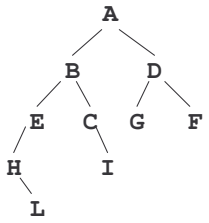
$$pos(T_x) = \begin{cases} "" & \text{if x is an external node} \\ pos(T_{L(x)}) + pos(T_{R(x)}) + x & \end{cases}$$

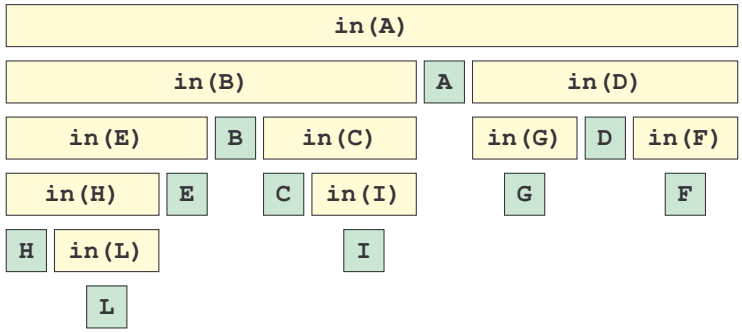




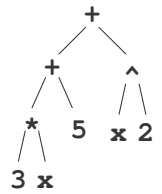
Inorder Traversal

$$in(T_x) = \begin{cases} \text{""} & \text{if x is an external node} \\ in(T_{L(x)}) + x + in(T_{R(x)}) & \end{cases}$$





Traversals กับ Expression Trees




```
preorder  = + + * 3 x 5 ^ x 2  -> prefix
inorder   = 3 * x + 5 + x ^ 2  -> infix
postorder = 3 x * 5 + x 2 ^ +  -> postfix
```


Traversal Templates

```
static void preorder(BinaryNode r) {
  if ( r != null ) {
    visit( r );
    preorder( r.left );
    preorder( r.right );
  }
}
```

```
static void inorder(BinaryNode r) {
  if ( r != null ) {
    inorder( r.left );
    visit( r );
    inorder( r.right );
  }
}
```

```
static void postorder(BinaryNode r) {
  if ( r != null ) {
    postorder ( r.left );
    postorder ( r.right );
    visit( r );
  }
}
```


ExpressionTree : toString( ) ก็คือ inorder

```
public String toString() {
  return toString(root);
}
private static String toString(BinaryNode r) {
  String s = "";
  if (r != null) {
    s  = "(" + toString(r.left);
    s += " " + r.element.toString();
    s += " " + toString(r.right) + ")";
  }
  return s;
}
```

inorder อยู่ตรงนี้

หมายเหตุ : toString ที่เคยเขียนก่อนหน้านี้ต่างกับที่เขียนข้างบนนี้เล็กน้อย

ExpressionTree : eval ใช้ postorder

```
public double eval(double x) {
  return eval(root, x);
}
private static double eval(BinaryNode r, double x) {
  double s = 0;
  if ( r != null ) {
    if (r.left == null && r.right == null) {
      if (r.element instanceof Double)
        s = ((Double)r.element).doubleValue();
      else
        s = x;
    } else {
      double s1 = eval(r.left, x);
      double s2 = eval(r.right, x);
      s = ((OP) r.element).eval(s1, s2);
    }
  }
  return s;
}
```

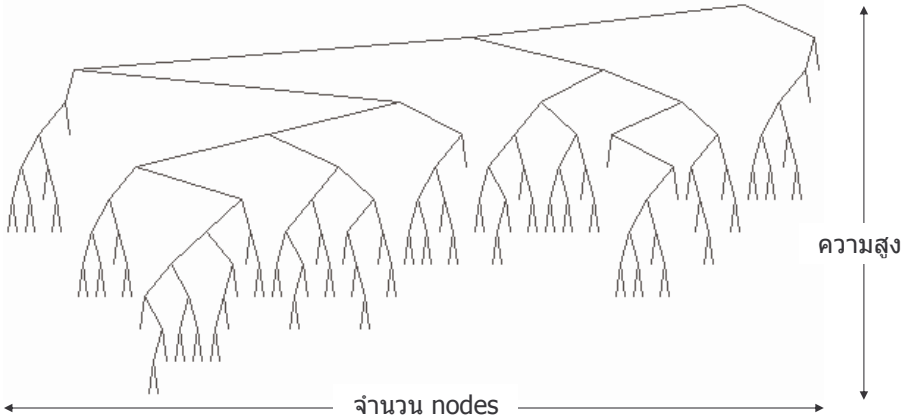
ใบ เป็นตัวแปร หรือไม่ก็ค่าคงตัว

postorder อยู่ตรงนี้

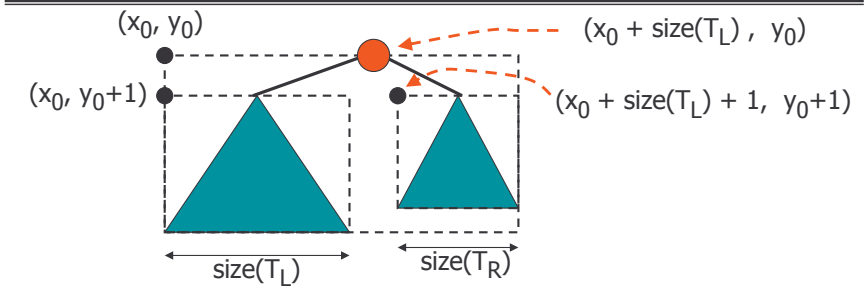
eval ตัวนี้อยู่ใน enum OP

การวาดต้นไม้

- มีตาราง (#nodes) × (height)
- วาง node ตามจุดตัดในตาราง
- จะวาง node ไหนไว้ที่ใด ?



การวาดต้นไม้




```
void drawTree(Graphics g, BinaryNode r, int x0, int y0) {
  if ( r != null ) {
    int sizeTL = size(r.left);
    drawNode(g, r.element, x0 + sizeTL, y0);
    drawTree(g, r.left, x0, y0 + 1);
    drawTree(g, r.right, x0 + sizeTL + 1, y0 + 1);
  }
}
```

ไม่ได้แสดงการวาดกิ่ง

พิกัดต่าง ๆ ในที่นี้ เป็นพิกัดของตารางการวาง nodes , หนึ่งหน่วยอาจกินหลาย pixels บนจอภาพ (ต้องคำนวณเสริม)

การบ้าน : ExpressionTree.toCanvas()

- เพิ่ม toCanvas() ให้กับ ExpressionTree ที่จะคืน canvas ที่สามารถนำไปใส่ใน Frame หรือ Panel เพื่อแสดงรูปร่างของต้นไม้

```
public class ExpressionTree {
  ...
  public Canvas toCanvas() {
    return new Canvas() {
      public void paint(Graphics g) { drawTree(g); }
    };
  }
  private void drawTree(Graphics g) {
    drawTree(g, root, 1, 1);
  }
  ...
```

ที่แสดงนี้เป็นแค่โครง คงต้องเขียนปรับปรุงอีก