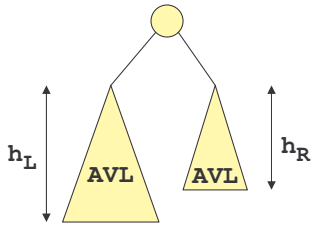
2110211 โครงสร้างข้อมูลเบื้องต้น

AVL Tree

สมชาย ประสิทธิ์จูตระกูล

AVL Trees

- binary search tree มีความสูง $n - 1 \le h \le \lfloor \log_2 n \rfloor$
- AVL = BST + กฎ (height-balanced)

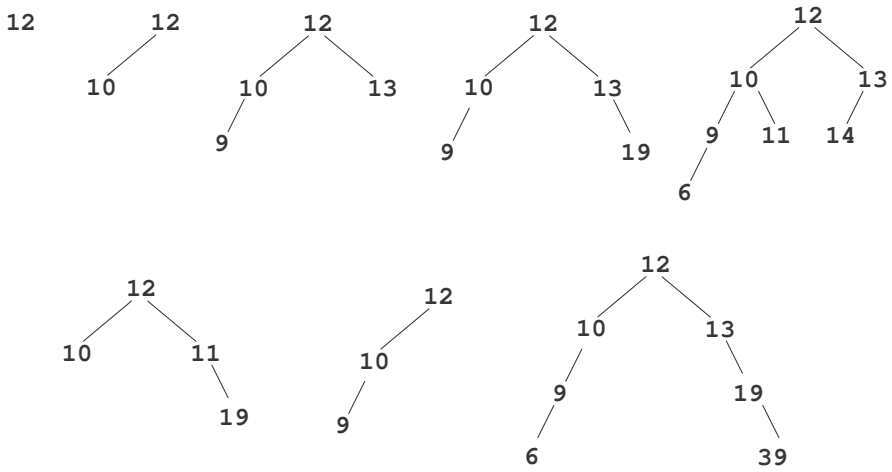


$$\left| h_L - h_R \right| \le 1$$

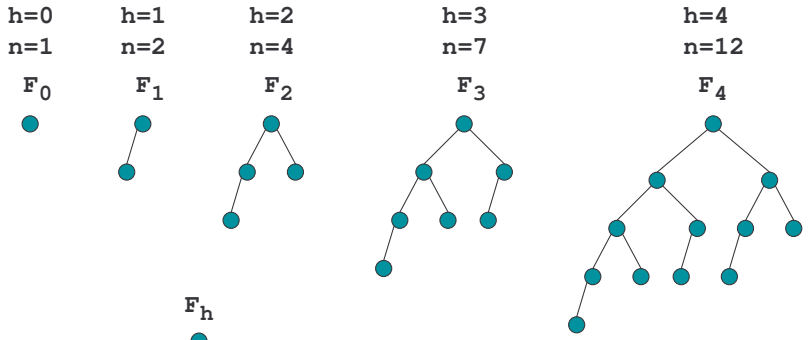
ต้นไม้ย่อยทุกต้นต้องเป็นไปตามกฎ

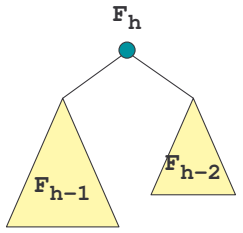
AVL : Adelson-Velskii and Landis

ต้นใดเป็น AVL ?



มาลองวาด AVL ที่สูง h โดยใช้ nodes น้อยสุด





$|F_h| = 1 + |F_{h-1}| + |F_{h-2}|$

Fibonacci Tree

ความสูงของต้นไม้ Fibonacci

$$|F_h| = 1 + |F_{h-1}| + |F_{h-2}|$$

$$n_h = 1 + n_{h-1} + n_{h-2} \quad h \geq 2,\ n_0 = 1, n_1 = 2$$

$$n_h = \alpha_1 \phi^h + \alpha_2 \hat{\phi}^h - 1,\ \phi = 1.618..., \hat{\phi} = -0.618$$

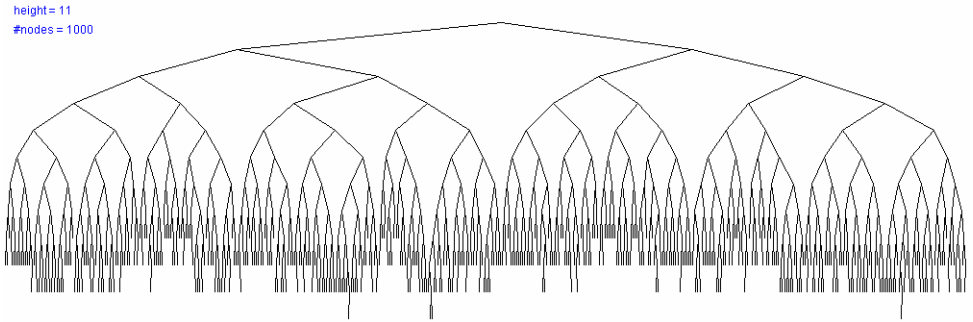
$$n_h \approx \alpha_1 \phi^h$$

$$h \approx \frac{1}{\log_2 \phi}\left(\log_2 n_h\right)$$

$$h \approx 1.44\left(\log_2 n_h\right)$$

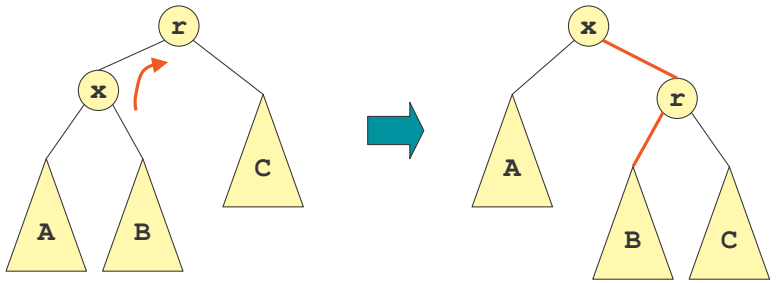
สรุป :
AVL tree ที่มี n nodes
สูงไม่เกิน 1.44 $\log_2 n$

Randomized AVL Tree



ทำอย่างไรให้เป็นไปตามกฎของ AVL

- อาศัยการหมุน (rotation)
- การหมุน node ใน bst ใดๆ ยังคงรักษาความเป็น bst



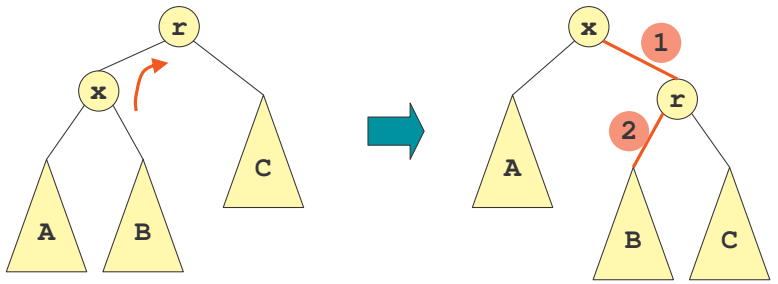
หมุนลูกซ้ายของ `r`
`rotateLeftChild(r)`

หลังการหมุน ต้นไม้ที่เคยมี `r` เป็นราก ก็เปลี่ยนรากเป็นลูกซ้ายของ `r` แทน

RotateLeftChild(r)

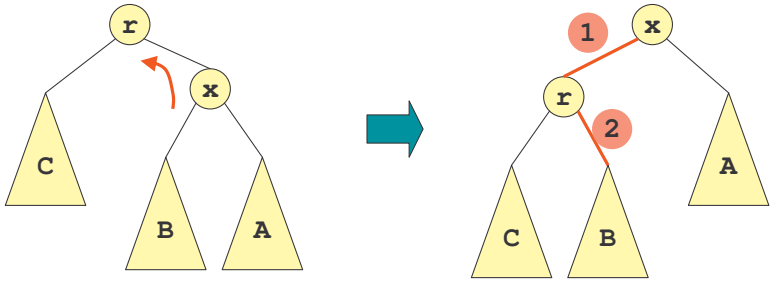
```
public Node rotateLeftChild(Node r) {
  assert r.left != null;
  Node newRoot = r.left;
  r.left = newRoot.right;      ← 2
  newRoot.right = r;           ← 1
  return newRoot;
}
```

O(1)



RotateRightChild(r)

```
public Node rotateRightChild(Node r) {
  assert r.right != null;
  Node newRoot = r.right;
  r.right = newRoot.left;      ← 2
  newRoot.left = r;            ← 1
  return newRoot;
}                                        O(1)
```



rebalance

- rebalance จะปรับต้นไม้ (อาศัยการหมุน) เมื่อพบว่าผิดกฎ
- ทุก node จะเก็บ height ของต้นไม้ย่อยที่ node นั้นเป็นราก
- rebalance อาศัย height ในการตรวจว่าผิดกฎหรือไม่
- เราเรียก rebalance ทุกครั้งที่มีการเพิ่มหรือลบข้อมูล

```
private Node add(Node r, Object e) {
  if (r == null) {
    r = new Node(e, null, null);
    ++size;
  } else {
    int cmp = ((Comparable) r.element).compareTo(e);
    if (cmp > 0) r.left = add(r.left, e);
    else if (cmp < 0) r.right = add(r.right, e);
  }
  r = rebalance(r);   ← ต่างกับของ bst แค่บรรทัดนี้เท่านั้น
  return r;
}
```


rebalance หลังการลบ

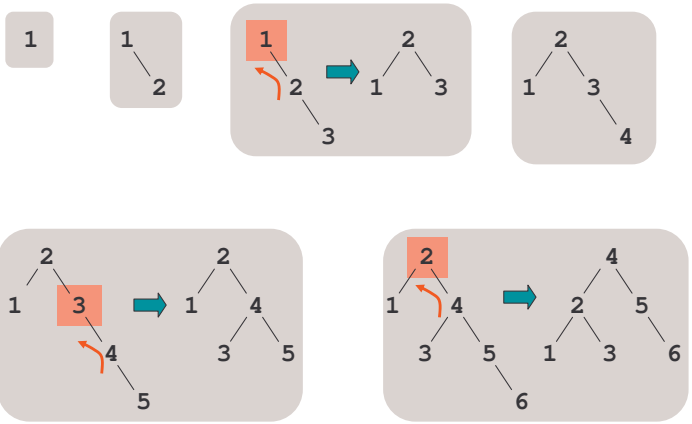
```
private Node remove(Node r, Object e) {
  if (r != null) {
    int cmp = ((Comparable) r.element).compareTo(e);
    if (cmp > 0) {
      r.left = remove(r.left, e);
    } else if (cmp < 0) {
      r.right = remove(r.right, e);
    } else {
      if (r.left == null || r.right == null) {
        r = (r.left == null ? r.right : r.left);
        --size;
      } else {
        r.element = getMinNode(r.right).element;
        r.right = remove(r.right, r.element);
      }
    }
    r = rebalance(r);   ← ต่างกับของ bst แค่บรรทัดนี้เท่านั้น
  }
  return r;
}
```


Node ต้องเก็บ height

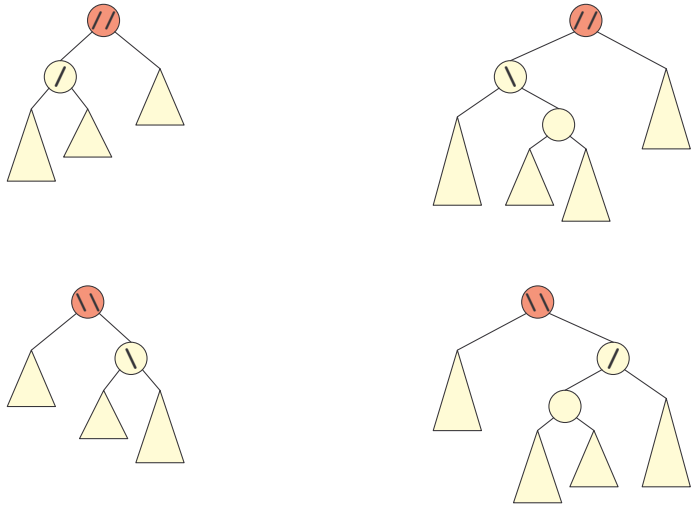
```
public class AVLSet implements Set {
  private static class Node {
    Object element;
    Node left;
    Node right;
    int height;
    Node(Object e, Node l, Node r) {
      this.element = e;
      this.left = l;
      this.right = r;
    }
    static int height(Node n) {
      return (n == null ? -1 : n.height);
    }
    void setHeight() {
      height = 1 + Math.max(height(left), height(right));
    }
    int balanceValue() {
      return height(right) - height(left);
    }
  }
```

การทำงานของ rebalance

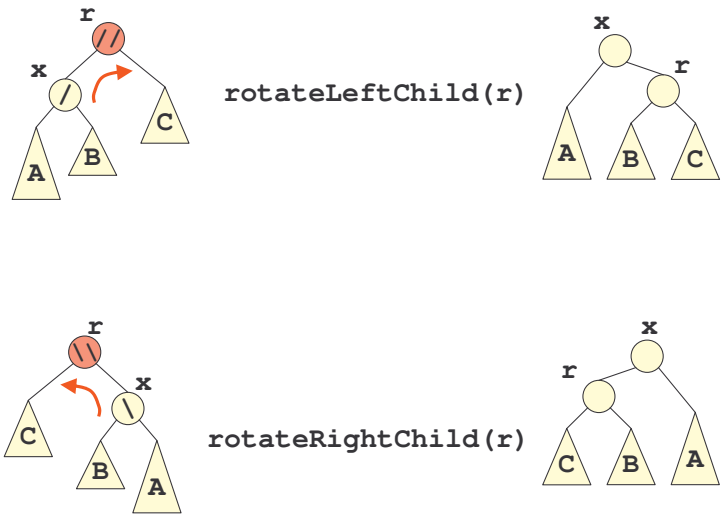
- ดูจากตัวอย่าง : เพิ่ม 1, 2, 3, 4, 5, 6, 7



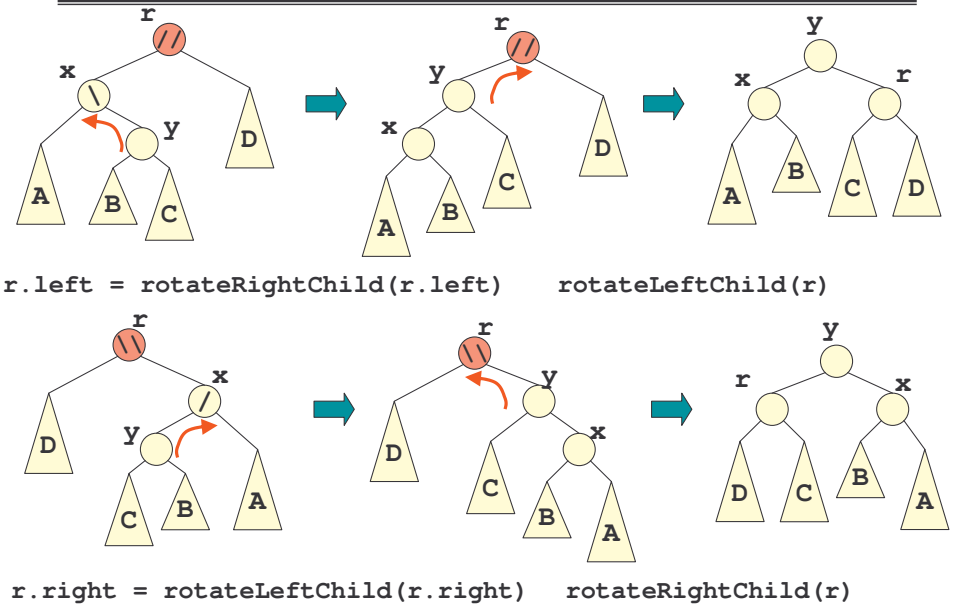
การ rebalance มี 4 กรณีย่อย



rebalance แบบหมุนลูกขึ้นเป็นราก

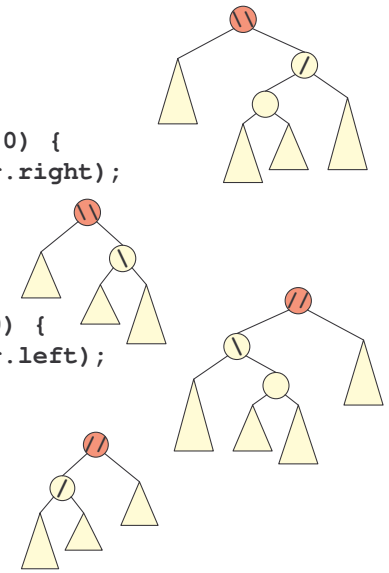


rebalance แบบหมุนหลานขึ้นเป็นราก



rebalance

```
private Node rebalance(Node r) {
  if (r != null) {
    int balance = r.balanceValue();
    if (balance > 1) {
      if (r.right.balanceValue() < 0) {
        r.right = rotateLeftChild(r.right);
      }
      r = rotateRightChild(r);
    } else
    if (balance < -1) {
      if (r.left.balanceValue() > 0) {
        r.left = rotateRightChild(r.left);
      }
      r = rotateLeftChild(r);
    }
    if (r != null) r.setHeight();
  }
  return r;
}
```



การบ้าน

- จงวิเคราะห์ว่า rebalance( ) ใช้เวลาเท่าใด ?
- การเพิ่ม rebalance ใน add และ remove ทำให้ add และ remove ใช้เวลาเท่าใด ?
- rotateLeftChild กับ rotateRightChild ที่เขียนให้ยังไม่มีการปรับความสูงหลังการหมุนเลย จงแก้ไข
- ลองสร้างต้นไม้ AVL ด้วยข้อมูลสุ่มสักหนึ่งล้านตัว แล้วดูสิว่า AVL ต้นนี้มีความสูงเท่าใด ?
- แก้ไข BSTSet และ AVLSet ให้สามารถเขียนคลาส AVLSet ใหม่ที่ extends จาก BSTSet ได้

การบ้าน : สร้าง map ด้วย AVL tree

- Map เป็นเสมือนที่เก็บคู่ลำดับ (key, value)
- ไม่มี key ซ้ำกันใน map
- การค้นจะให้ค่าของ key แล้วได้ value ที่คู่กัน คืนกลับมา
- จงสร้าง AVLMap ที่ทำเป็น map และ extends จาก AVLSet

```
public class AVLMap extends AVLSet implements Map
```

```
public interface Map {
  public int size();
  public boolean isEmpty();
  public boolean containsKey(Object key);
  public boolean containsValue(Object value);
  public Object get(Object key);
  public Object put(Object key, Object value);
  public void remove(Object key);
}
```

ข้อกำหนดของแต่ละเมท็อดหาอ่านได้จาก javadoc ไม่ต้องสร้างให้ครบเท่า java.util.Map เอาเท่าที่แสดงข้างบนนี้ก็พอ