ต้นไม้ค้นหาแบบทวิภาค

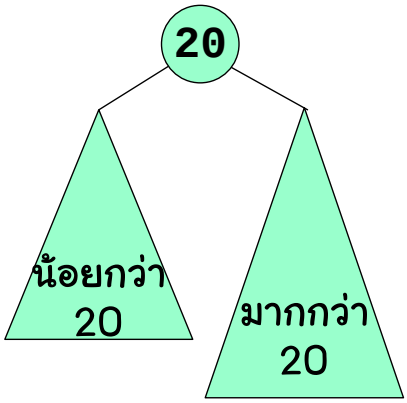
(Binary Search Tree)

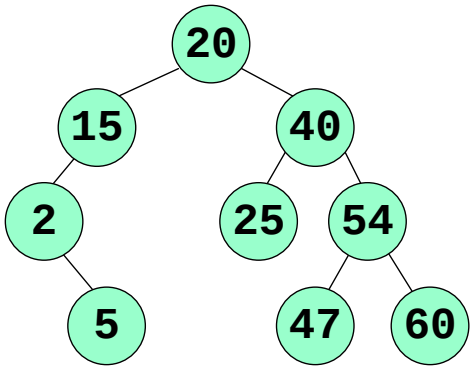
หัวข้อ

- นิยามต้นไม้ค้นหาแบบทวิภาค
- โครงสร้างของต้นไม้ค้นหาแบบทวิภาค
- บริการต่าง ๆ
  - การค้นหาข้อมูล ตัวน้อยสุด ตัวมากสุด
  - การเพิ่ม
  - การลบ
  - การเรียงลำดับข้อมูลโดยใช้ต้นไม้ค้นหาแบบทวิภาค
- การสร้างเซตและคอลเล็กชัน ด้วยต้นไม้ค้นหาแบบทวิภาค

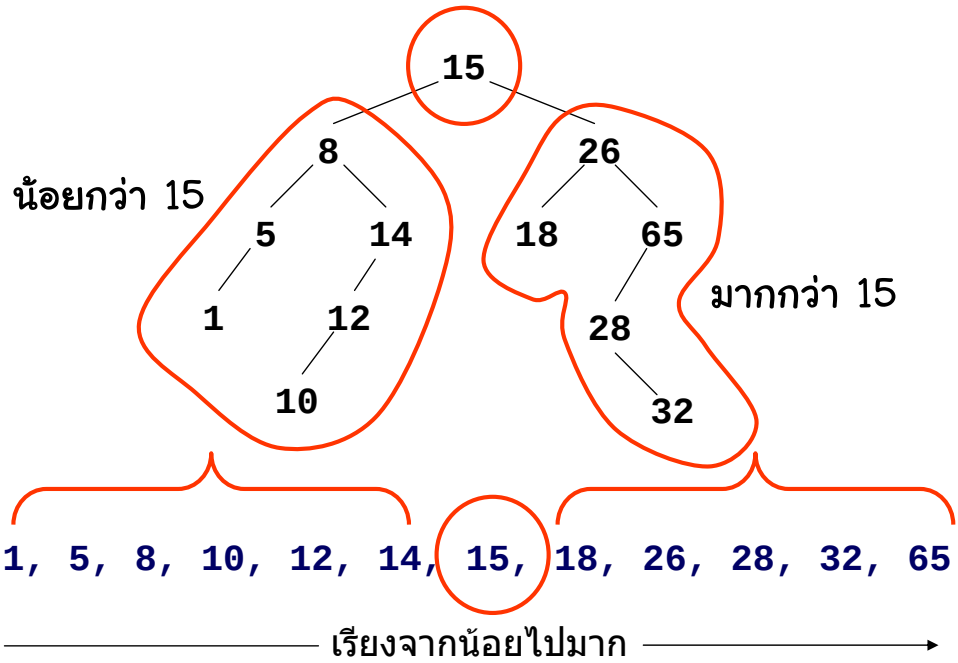
ต้นไม้ค้นหาแบบทวิภาค

- เป็นต้นไม้แบบทวิภาค
- เก็บข้อมูลตามปม
- ข้อมูลในต้นไม้ย่อยทางซ้ายต้องน้อยกว่าข้อมูลที่ราก
- ข้อมูลในต้นไม้ย่อยทางขวาต้องมากกว่าข้อมูลที่ราก
- ต้นไม้ย่อยทุก ๆ ต้นต้องเป็น binary search tree ด้วย

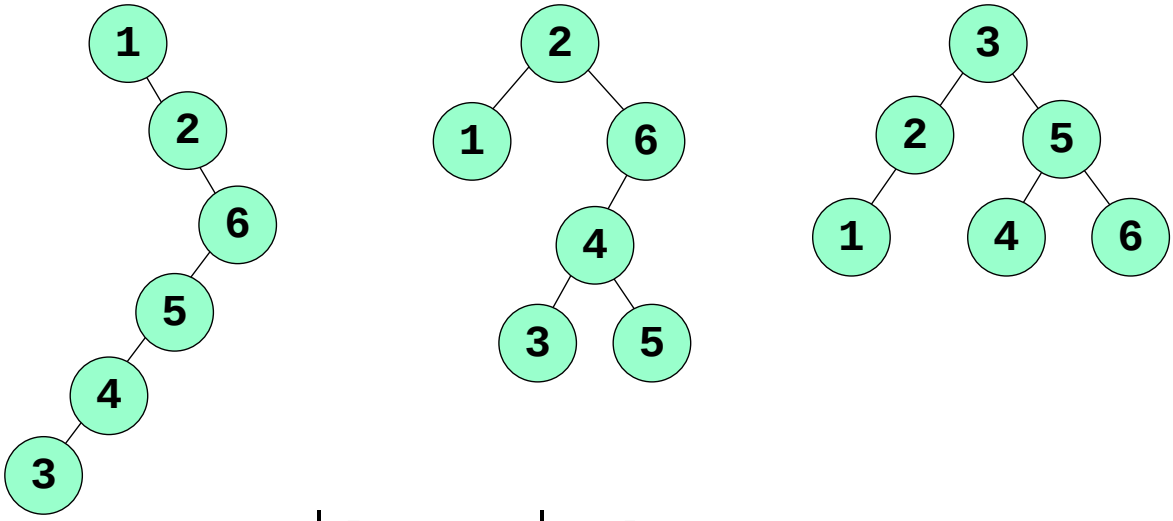




การแวะผ่านแบบตามลำดับ



ข้อมูลชุดเดียวกันเก็บได้หลายแบบ



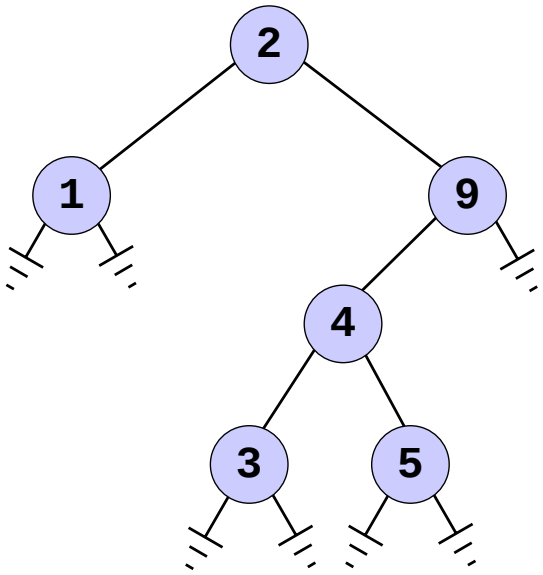
$$\lfloor \log_2 n \rfloor \le h \le n-1$$

class BSTree

```
public class BSTree extends BinaryTree {
  int size;

  public BSTree() {}
  public int size() { return size; }
  public boolean isEmpty() { return size == 0; }

  int compare(Object a, Object b) {
    return ((Comparable)a).compareTo(b);
  }

  public Object get(Object e) {...}
  public Object getMin() {...}
  public Object getMax() {...}
  public void add(Object e) {...}
  public void remove(Object e) {...}
  public static treeSort(Object[] data) {...}
  ...
```

การค้นหาข้อมูล

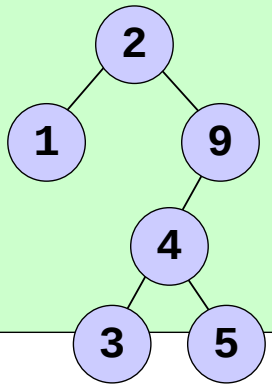
- ใช้การแวะผ่านต้นไม้ ค่อย ๆ เปรียบเทียบ
- ใช้กฎการจัดเก็บช่วยในการค้น



$O(h)$

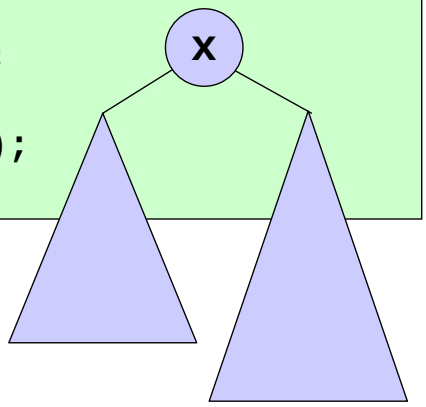
get : การค้นหาข้อมูล

```
public class BSTree extends BinaryTree {
  ...
  public Object get(Object e) {
    Node node = getNode(root, e);
    return node == null ? null : node.element;
  }
  Node getNode(Node r, Object e) {
    while (r != null) {
      int cmp = compare(e, r.element);
      if (cmp == 0) return r;
      if (cmp < 0)
        r = r.left;
      else
        r = r.right;
    }
    return null;
  }
```



การค้นหาข้อมูลแบบเวียนเกิด

```
public class BSTree extends BinaryTree {
  ...
  Node getNode(Node r, Object e) {
    if (r == null) return null;
    int cmp = compare(e, r.element);
    if (cmp == 0) return r;
    if (cmp < 0)
      return getNode(r.left, e);
    else
      return getNode(r.right, e);
  }
```

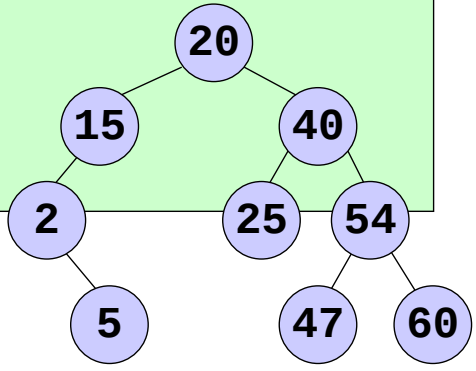


getMin : การค้นหาข้อมูลตัวน้อยสุด

- ปมใดมีลูกทางซ้าย ย่อมมีข้อมูลที่น้อยกว่า
- หาตัวน้อยสุด ทำได้โดยเริ่มที่รากแล้วลงไปทางซ้าย จนกว่าจะพบปมที่ไม่มีลูกซ้าย

```
public Object getMin() {
  Node r = root;
  if (r == null) return null;
  while (r.left != null) {
    r = r.left;
  }
  return r.element;
}
```

$O(h)$

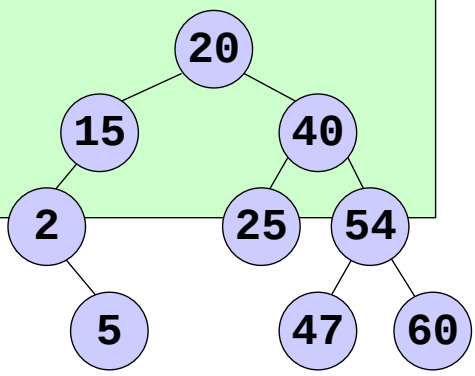


getMax : การค้นหาข้อมูลตัวมากสุด

- ปมใดมีลูกทางขวา ย่อมมีข้อมูลที่มากกว่า
- หาตัวมากสุด ทำได้โดยเริ่มที่รากแล้วลงไปทางขวา จนกว่าจะพบปมที่ไม่มีลูกขวา

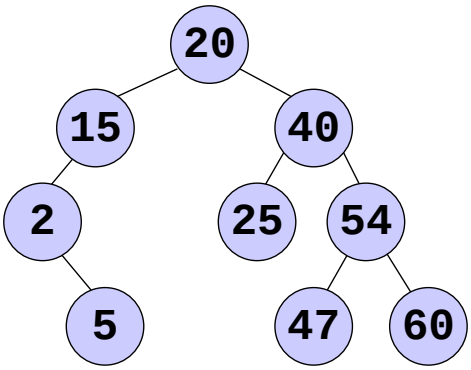
```
public Object getMax() {
  Node r = root;
  if (r == null) return null;
  while (r.right != null) {
    r = r.right;
  }
  return r.element;
}
```

$O(h)$



การเพิ่มข้อมูล

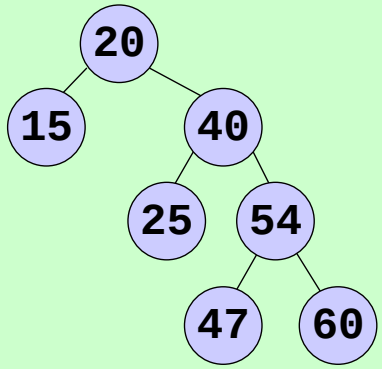
- สร้างปมใหม่, เพิ่มเข้าในต้นไม้
- เพิ่มเป็นใบใหม่ ณ ตำแหน่งที่ได้จากการค้น



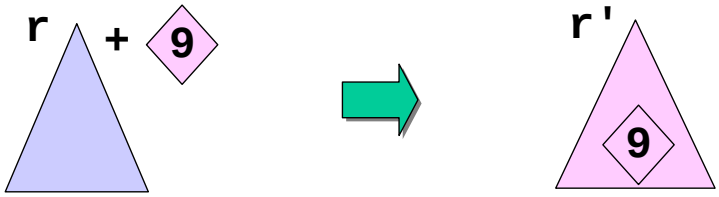
$O(h)$

add : การเพิ่มข้อมูล

```
public void add(Object e) {
  Node newNode = new Node(e, null, null);
  if (root == null) root = newNode;
  else {
    Node p = null, r = root;
    while( r != null ) {
      int cmp = compare(e, r.element);
      if      (cmp < 0) {p = r; r = r.left;}
      else if (cmp > 0) {p = r; r = r.right;}
      else return;
    }
    if (compare(e, p.element) < 0)
      p.left = newNode;
    else
      p.right = newNode;
  }
  ++size;
}
```

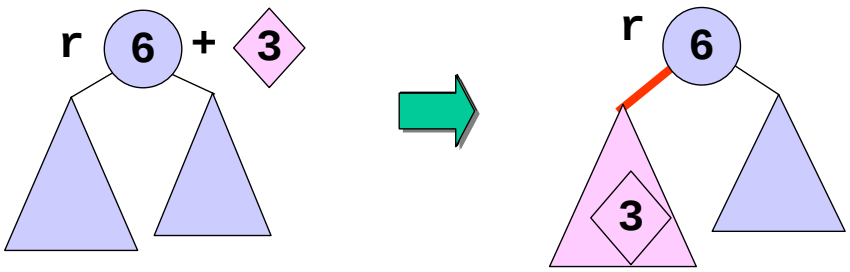


การเพิ่มข้อมูลแบบเวียนเกิด




```
if (r == null) return new Node(e, null, null)
```


```
if (compare(e,r.element)<0) r.left = add(r.left,e);
```



add : แบบเวียนเกิด

```
public void add(Object e) {
  root = add(root, e);
}
Node add(Node r, Object e) {
  if (r == null) {
    r = new Node(e, null, null);
    ++size;
  } else {
    int cmp = compare(e, r.element);
    if (cmp < 0)
      r.left = add(r.left, e);
    else if (cmp > 0)
      r.right = add(r.right, e);
  }
  return r;
}
```

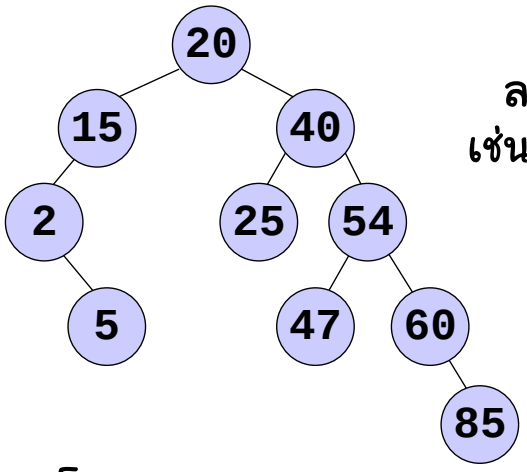

ลักษณะของต้นไม้ขึ้นกับลำดับการเพิ่ม

1, 2, 6, 3, 5 **2, 1, 5, 3, 6**

ความสูงของต้นไม้ขึ้นกับลำดับ
ของข้อมูลที่เพิ่มใส่ต้นไม้

การลบข้อมูล

- ค้นหาปมที่เก็บข้อมูลที่ต้องการลบ
- ลบปมที่เก็บข้อมูลนั้น หรือลบข้อมูลในปมนั้น



ลบปมที่มี 2 ลูก
เช่น ต้องการลบ 40

$O(h)$

ลบใบ
เช่น ต้องการลบ 5

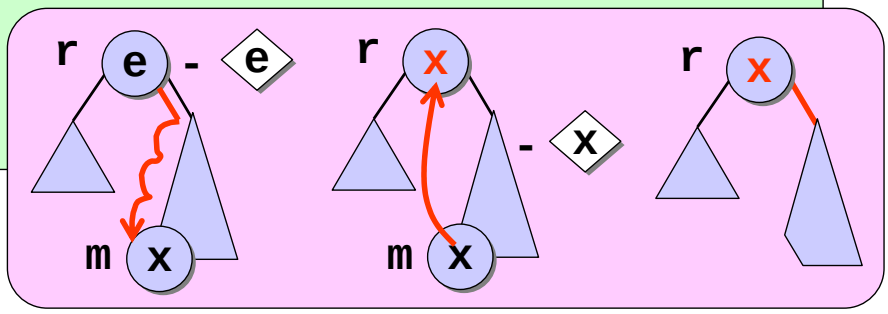
ลบปมที่มีลูกเดียว
เช่น ต้องการลบ 60

remove : แบบเวียนเกิด

```
public void remove(Object e) {
  root = remove(root, e);
}
Node remove(Node r, Object e) {
  if (r == null) return r;
  int cmp = compare(e, r.element);
  if (cmp < 0) {
    r.left = remove(r.left, e);
  } else if (cmp > 0) {
    r.right = remove(r.right, e);
  } else {
    // พบแล้ว ลบที่นี่



  }
  return r;
}
```

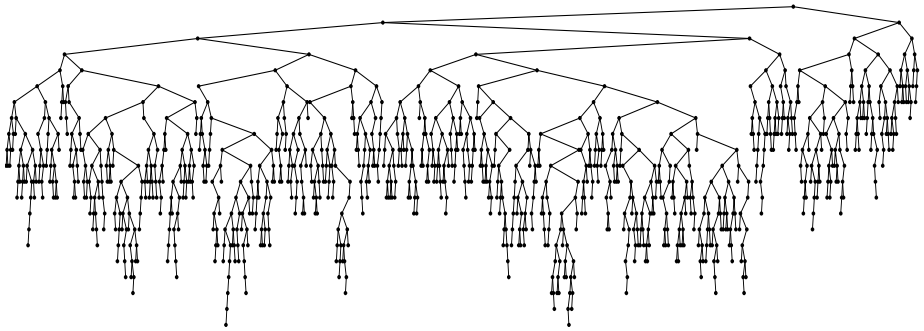
remove : แบบเวียนเกิด

```
Node remove(Node r, Object e) {
  ...
  } else {
    if (r.left == null || r.right == null) {
      r = (r.left == null ? r.right : r.left);
      --size;
    } else {
      Node m = r.right;
      while (m.left != null) m = m.left;
      r.element = m.element;
      r.right = remove(r.right, m.element);
    }
  }
  return r;
}
```



ต้นไม้ BSTree ที่สร้างจากข้อมูลสุ่ม

- ต้นไม้ที่เก็บข้อมูล n ตัว
- ช่วงของความสูง : $\lfloor \log_2 n \rfloor \le h \le n-1$
- ถ้าสร้างจากข้อมูลสุ่ม สามารถวิเคราะห์ได้ว่า
  - ความลึกเฉลี่ยของปมภายใน $\approx 1.39 \log_2 n$
  - ความลึกเฉลี่ยของ null $\approx 2 + 1.39 \log_2 n$
  - ความสูง (ความลึกของใบล่างสุด) $\approx 2.99 \log_2 n$



Devroye, L. 1986. A note on the height of binary search trees. *J. ACM 33*, 489–498.

เวลาการทำงานของการเพิ่ม ลบ ค้น

- get, getMin, getMax, add, remove : O(h)
- ต้นไม้มีความสูงในช่วง $\lfloor \log_2 n \rfloor \leq h \leq n - 1$
- กรณีเร็วสุด (ต้นไม้เตี้ยสุด) : O(log n)
- กรณีช้าสุด (ต้นไม้สูงสุด) : O(n)
- กรณีเฉลี่ย (เมื่อต้นไม้สร้างจากข้อมูลสุ่ม) : O(log n)

การเรียงลำดับข้อมูลแบบต้นไม้

- นำข้อมูลทั้งหมด มาสร้างต้นไม้ค้นหาแบบทวิภาค
- แวะผ่านต้นไม้นี้แบบตามลำดับ

2, 1, 5, 3, 6

1 2 3 5 6

treeSort : การเรียงลำดับข้อมูล

```
public class BSTree extends BinaryTree {
  ...
  public static void treeSort(final Object[] data) {
    BSTree t = new BSTree();
    for (int i=0; i<data.length; i++) {
      t.add(data[i]);
    }
    t.inOrder(new Visitor() {
                int k = 0;
                public void visit(Object e) {
                  data[k++] = e;
                } } );
  }
  ...
```

เพิ่ม n ครั้งใช้เวลา O(n log n)

แวะผ่าน n ปม ใช้เวลา O(n)

$O(n \log n)$

การสร้างเซตด้วย BSTree

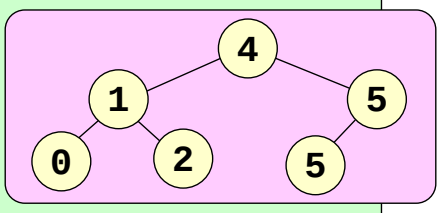
```
public class BSTSet implements Set {
  protected BSTree tree = new BSTree();
  public int size() {
    return tree.size();
  }
  public boolean isEmpty() {
    return tree.isEmpty();
  }
  public boolean contains(Object e) {
    return tree.get(e) != null;
  }
  public void add(Object e) {
    tree.add(e);
  }
  public void remove(Object e) {
    tree.remove(e);
  }
}
```

BSTree ไม่เก็บตัวซ้ำอยู่แล้ว

การสร้างคอลเล็กชันด้วย BSTree

- ปรับให้ BSTree เก็บตัวซ้ำ โดยให้ไปเพิ่มทางซ้าย
- ถ้ามีการลบ ตัวซ้ำอาจอยู่ทางขวาก็ได้ (ก็ไม่เป็นไร)

```
Node add(Node r, Object e) {
  if (r == null) {
    r = new Node(e, null, null);
    ++size;
  } else {
    int cmp = compare(e, r.element);
    if (cmp <= 0)
      r.left = add(r.left, e);
    else
      r.right = add(r.right, e);
  }
  return r;
}
```



สรุป

- ต้นไม้ค้นหาแบบทวิภาคมีจัดเก็บข้อมูลโดยอาศัยการเปรียบเทียบความมากกว่าน้อยกว่าของข้อมูล
- สามารถลดปริมาณข้อมูลที่ต้องพิจารณาได้ทีละมากๆ ระหว่างการเพิ่ม ลบ และค้นหา
- เวลาการทำงานขึ้นกับลักษณะของต้นไม้
- โชคดีทำงานเร็ว O(log n), โชคร้ายทำงานช้า O(n)
- เป็นโครงสร้างพื้นฐานของโครงสร้างข้อมูลอื่น ๆ ที่ซับซ้อนและมีประสิทธิภาพกว่า