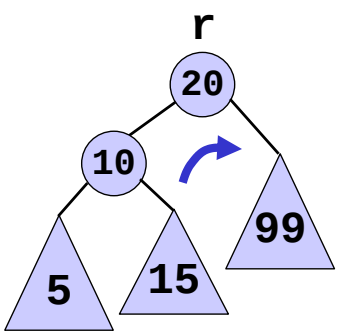
ต้นไม้ค้นหาแบบอื่น ๆ

หัวข้อ

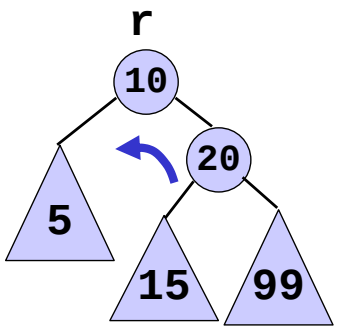
- ต้นไม้ทรีพ (treap)
- ต้นไม้บาน (splay tree)
- ต้นไม้ได้ดุล 2-3-4 (balanced 2-3-4 tree)
- ต้นไม้แดงดำ (red-black tree)

การหมุนปม

- ใช้หมุนปมให้สูงขึ้นหรือต่ำลง
- หมุนแล้วยังคงรักษาความเป็นต้นไม้ค้นหา
- ใช้เวลาคงตัว



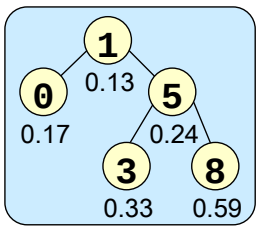
`rotateLeftChild(r)`



`rotateRightChild(r)`

ต้นไม้ทรีพ (Treap)

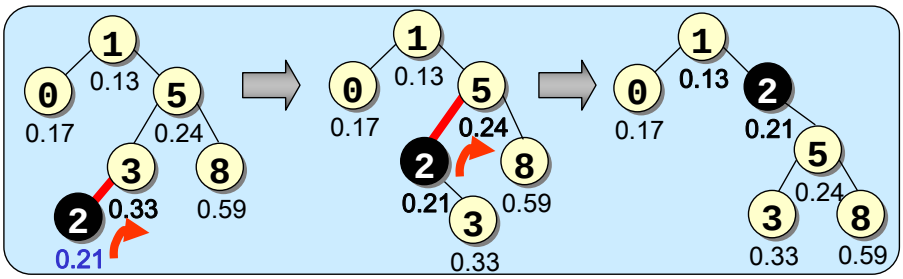
- นำแนวคิดของฮีปเพิ่มในต้นไม้ค้นหาแบบทวิภาค
- แต่ละปมมีข้อมูลสองตัว
  - ตัวข้อมูลจริง ของต้นไม้ค้นหาแบบทวิภาค
  - ข้อมูลเสริม เมื่อมองต้นไม้นี้เป็นฮีปแบบน้อยสุด
- ต้องรักษาโครงสร้างของต้นไม้ให้คงคุณสมบัติ
  - ข้อมูลจริงของต้นไม้ซ้ายน้อยกว่าราก ของต้นขวามากกว่าราก
  - ข้อมูลเสริมของปมพ่อต้องน้อยกว่าของลูก
  - ใช้การหมุนปม สลับความเป็นพ่อลูกได้



ข้อมูลเสริมของฮีปได้มาจาก
การสุ่มจำนวนจริงตอนสร้างปม

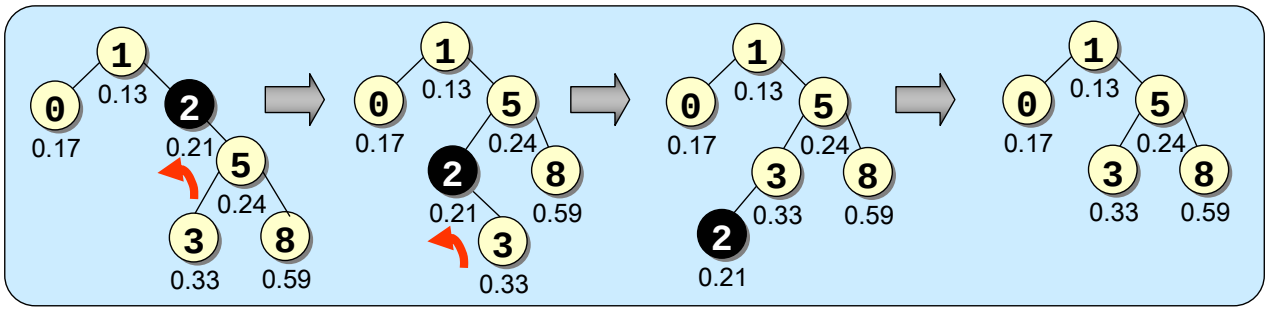
การเพิ่มข้อมูล

- ต้องการเพิ่ม x
  - เพิ่มปมใหม่เก็บ x เหมือนการเพิ่มในต้นไม้ค้นหาแบบทวิภาค
  - สุ่มจำนวนจริงเป็นข้อมูลเสริมกำกับปมใหม่
  - หมุนปม x ขึ้นไปเรื่อย ๆ จนกว่าจะไม่ผิดอันดับแบบฮีป



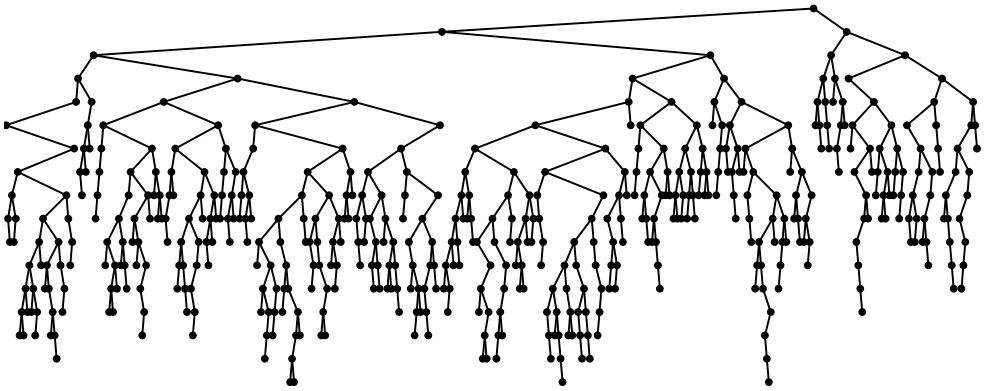
การลบข้อมูล

- ต้องการลบ x
  - ค้นปมที่เก็บ x
  - หมุนปมที่เก็บ x ลงไปเป็นใบ
  - ลบใบนั้นทิ้ง



สรุป : ต้นไม้ทรีพ

- ได้ต้นไม้สูงเฉลี่ยเป็น O(log n)
- การเพิ่ม ลบ ค้น ในกรณีเฉลี่ยใช้เวลา O(log n)

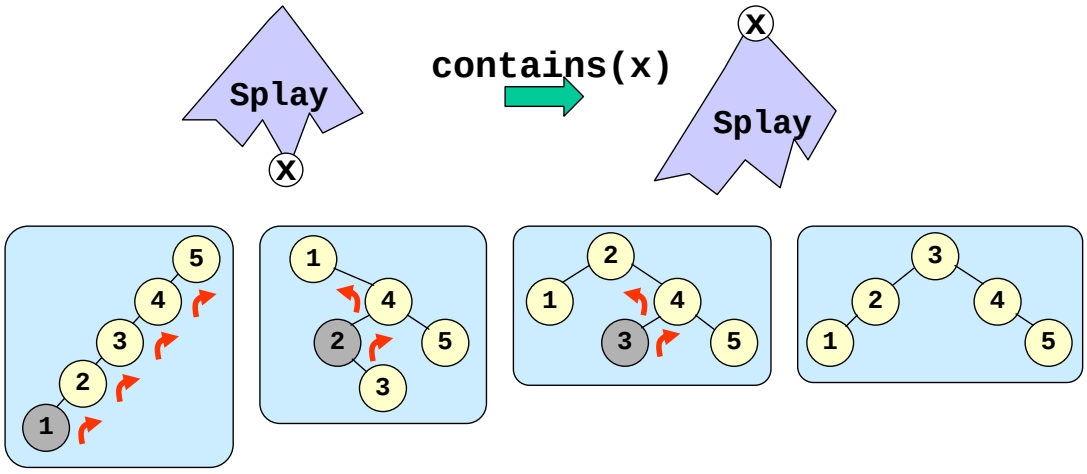


เพิ่มข้อมูลตามลำดับ 1,2,3,...,500

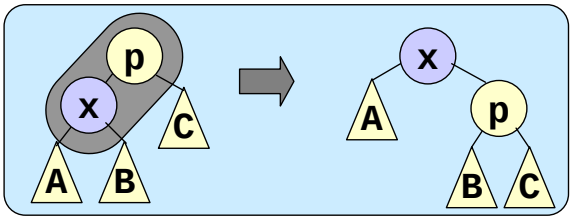
Treap = Binary search Tree + Heap

ต้นไม้บาน (Splay Tree)

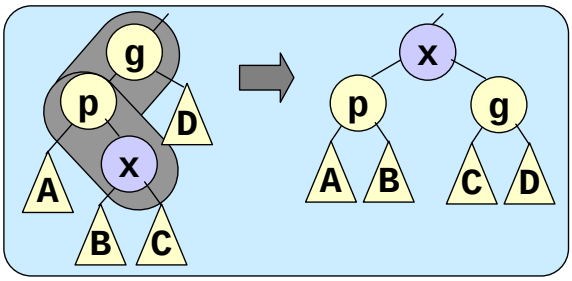
- เป็นต้นไม้ค้นหาแบบทวิภาคที่ปรับโครงสร้างตัวเอง
- จะปรับต้นไม้ทุกครั้งที่มีการเพิ่ม ลบ และค้น



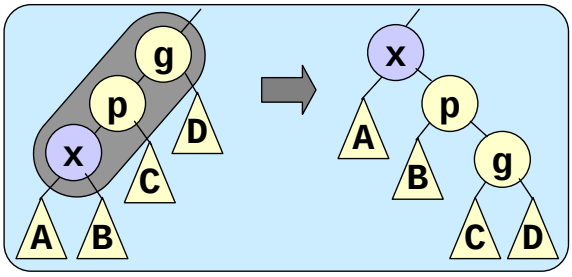
การปรับต้นไม้



zig

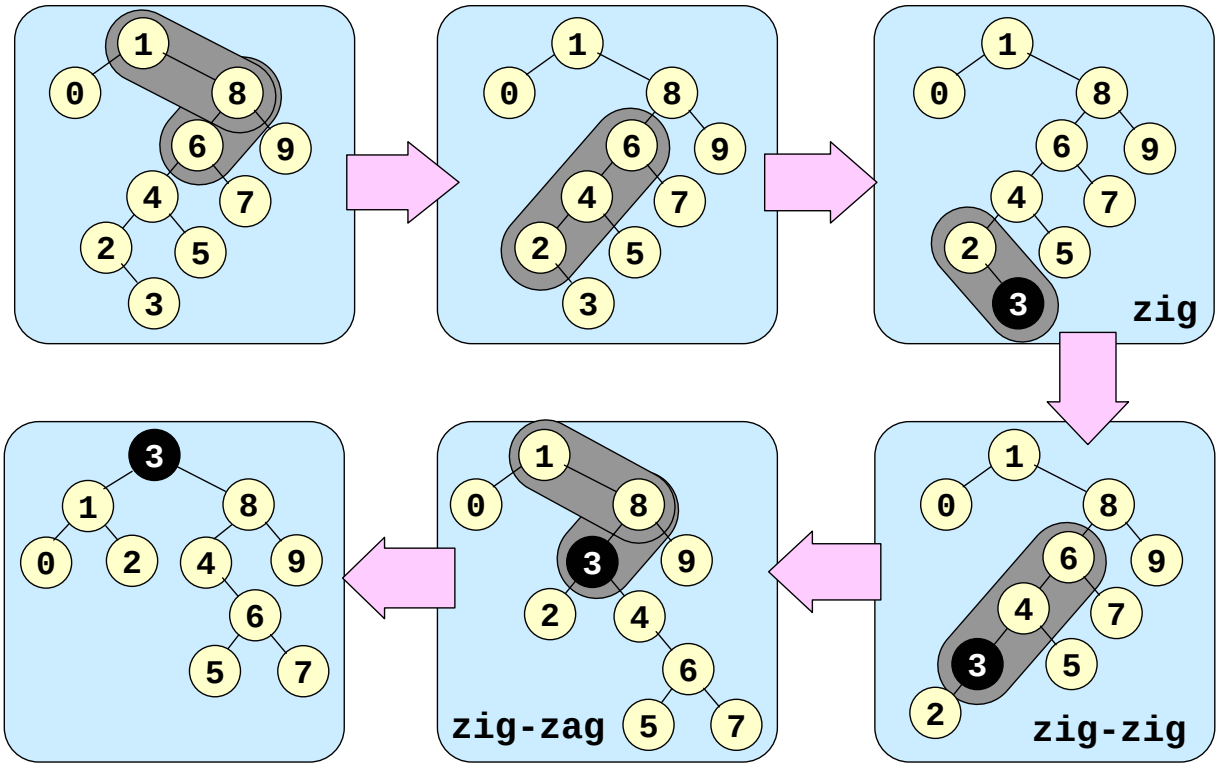


zig-zag



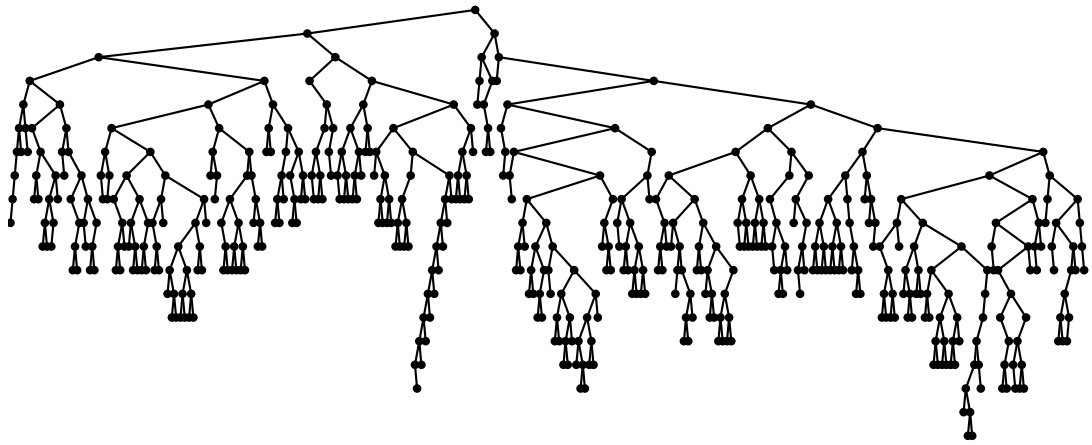
zig-zig

การค้นข้อมูล



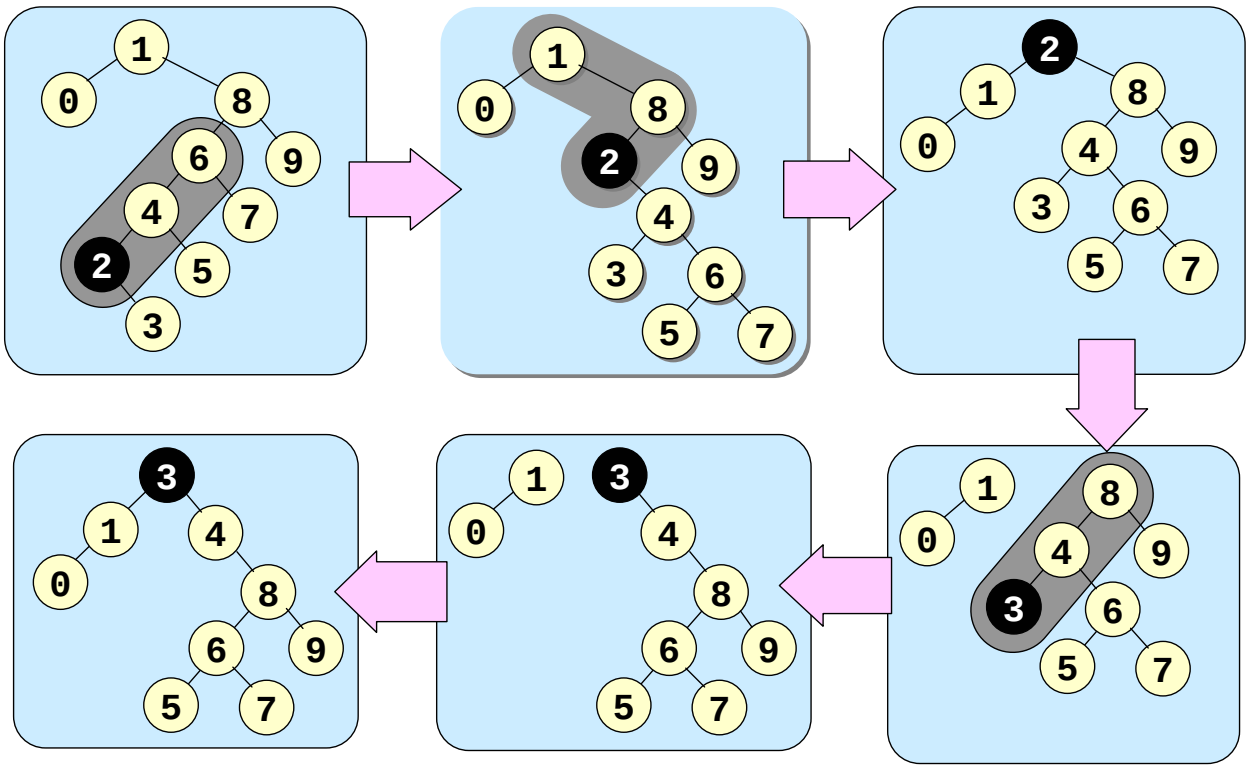
การเพิ่มข้อมูล

- เพิ่มตามปกติ (เหมือนของต้นไม้ค้นหาแบบทวิภาค)
- splay ปมข้อมูลใหม่ ขึ้นมาเป็นราก



เพิ่มข้อมูลตามลำดับ 1,2,3,...,500

การลบข้อมูล

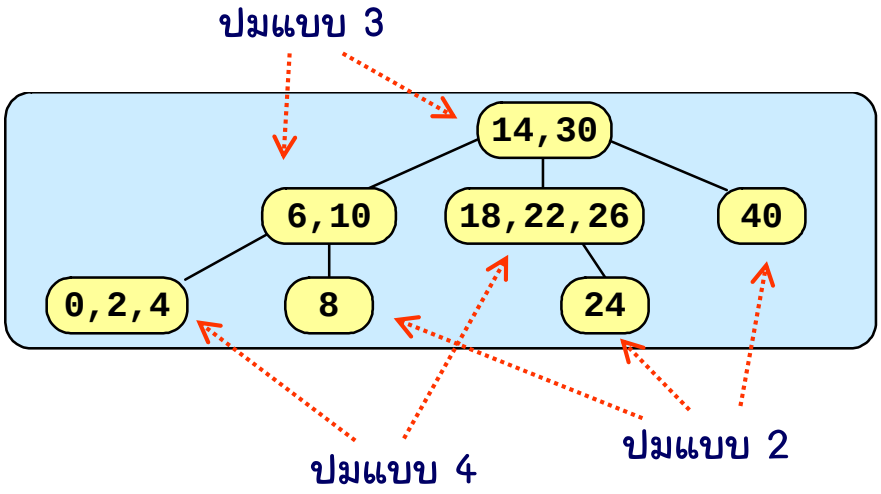


สรุป : ต้นไม้บาน

- ต้นไม้บานไม่ได้ประกันความสูงของต้นไม้
- ข้อมูลตัวที่ถูกค้นบ่อย ๆ จะอยู่บน ๆ จึงถูกค้นในอนาคตได้เร็ว
- ไม่ต้องข้อมูลเสริมใด ๆ ตามปม
- ประกันว่าเวลาสะสมของการให้บริการ add, remove, contains ต่าง ๆ จำนวน m ครั้ง เป็น O(m log n) โดยที่ n เป็นจำนวนข้อมูลของต้นไม้บาน
- แสดงว่าถัวเฉลี่ยต่อ operation คือ O(log n)

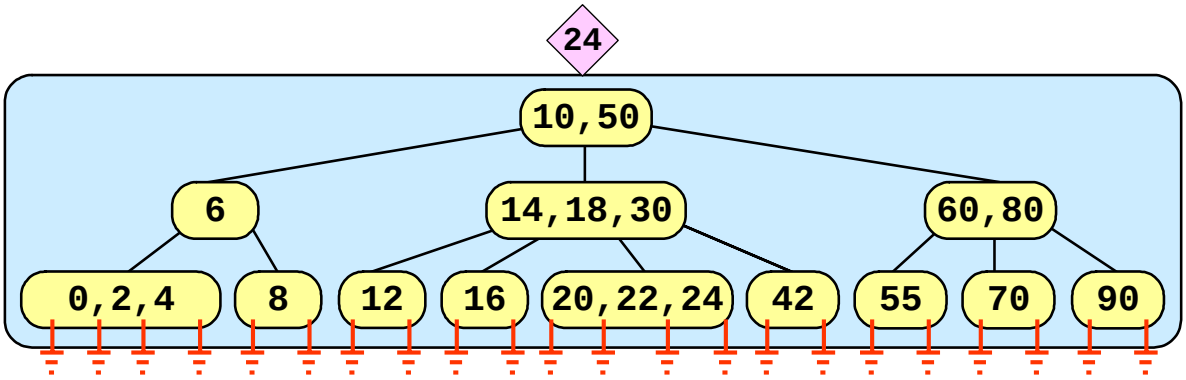
ต้นไม้ 2-3-4

- binary tree : ต้นไม้เตี้ยสุด $\log_2 n$
- m-ary tree : แต่ละปมมี m ลูก ต้นไม้เตี้ยสุด $\log_m n$
- ต้นไม้ 2-3-4 : แต่ละปมมี 2, 3, หรือ 4 ลูกได้
- ปมแบบ k เก็บข้อมูล k – 1 ตัว เรียงจากน้อยไปมาก



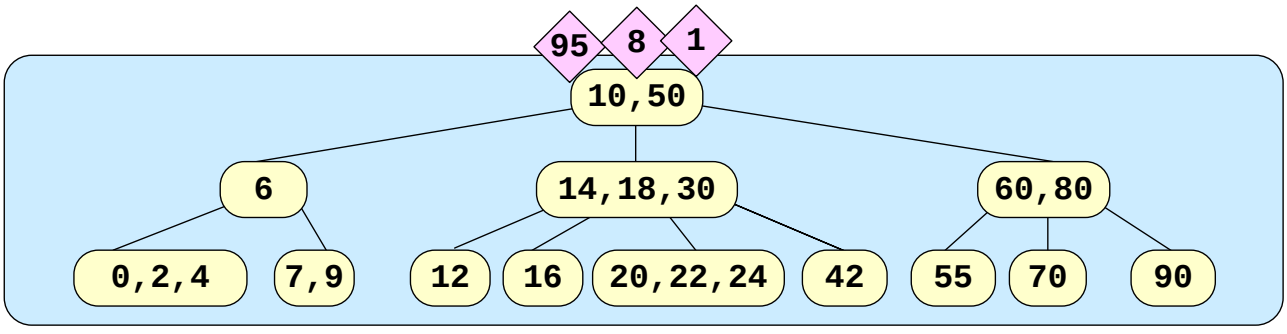
ต้นไม้ได้ดุล 2-3-4

- คือต้นไม้ 2-3-4 ที่ null อยู่ระดับเดียวกันหมด
- $\lfloor \log_4 n \rfloor \le h \le \lfloor \log_2 n \rfloor$



การเพิ่มข้อมูล

- ต้องการเพิ่ม x
  - หา x จนจบที่ใบ
  - ถ้าใบนั้นเป็นปมแบบ 2 ก็แทรก x ให้เป็นปมแบบ 3
  - ถ้าใบนั้นเป็นปมแบบ 3 ก็แทรก x ให้เป็นปมแบบ 4
  - ถ้าใบนั้นเป็นปมแบบ 4 แทรกไม่ได้ให้แตกปม และนำตัวกลางไปเพิ่มในปมระดับบน

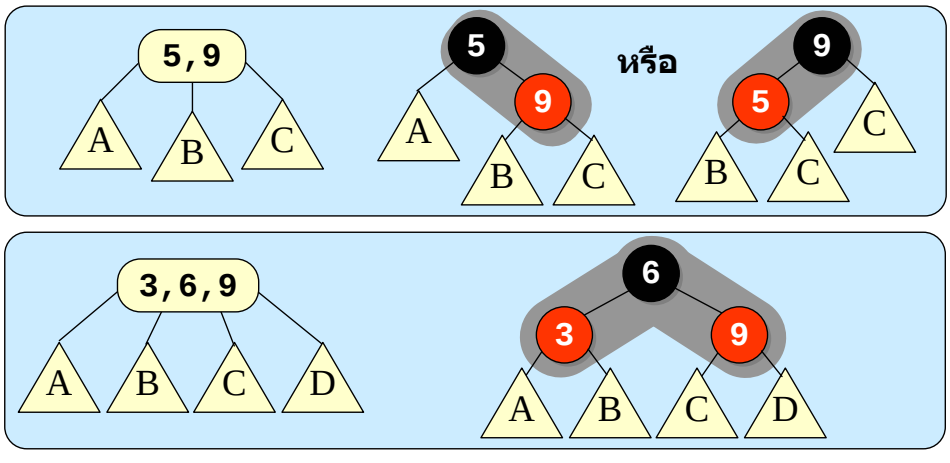


สรุป : ต้นไม้ได้ดุล 2-3-4

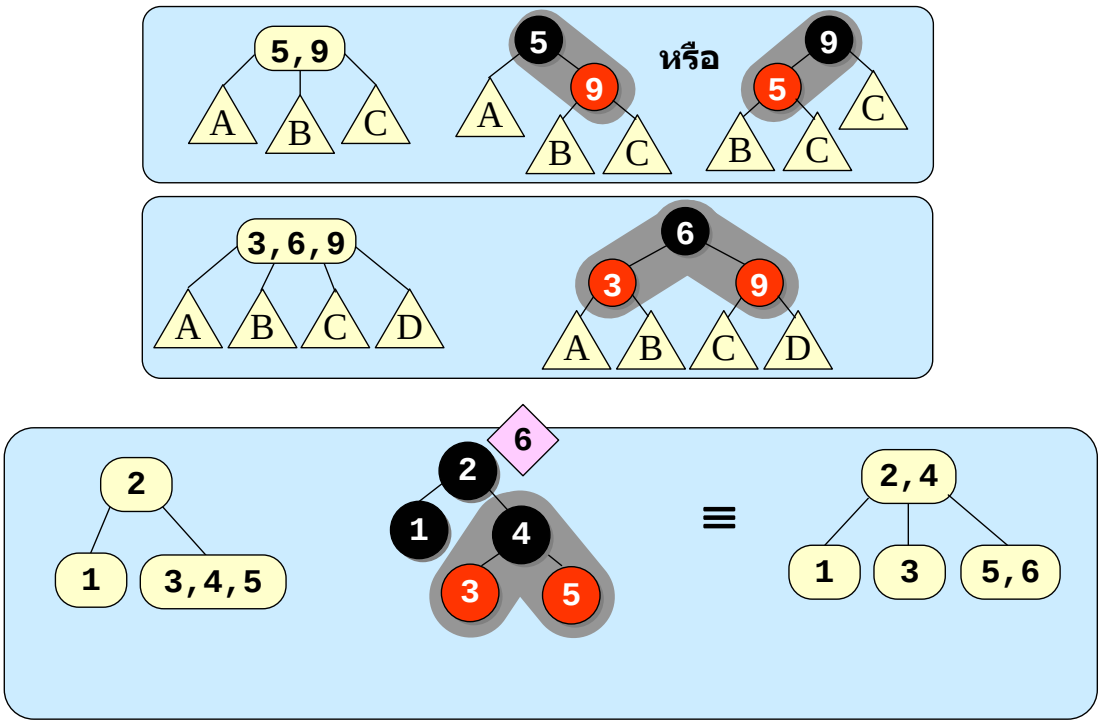
- เป็นต้นไม้ที่ประกันความสูง O(log n)
- เวลาในการค้น เพิ่ม ลบ เป็น O(log n)
- การลบจะซับซ้อนกว่า ทำได้โดยอาศัยการรวมปม ย้ายข้อมูลระหว่างปมญาติสนิท
- เป็นต้นไม้พื้นฐานของต้นไม้บี (B-tree) ซึ่งเป็นต้นไม้สำคัญในการการจัดเก็บฐานข้อมูลขนาดใหญ่ในฮาร์ดดิสก์ (แต่ละปมใน B-tree มีลูกได้เป็นร้อย)

ต้นไม้แดงดำ (Red-Black Tree)

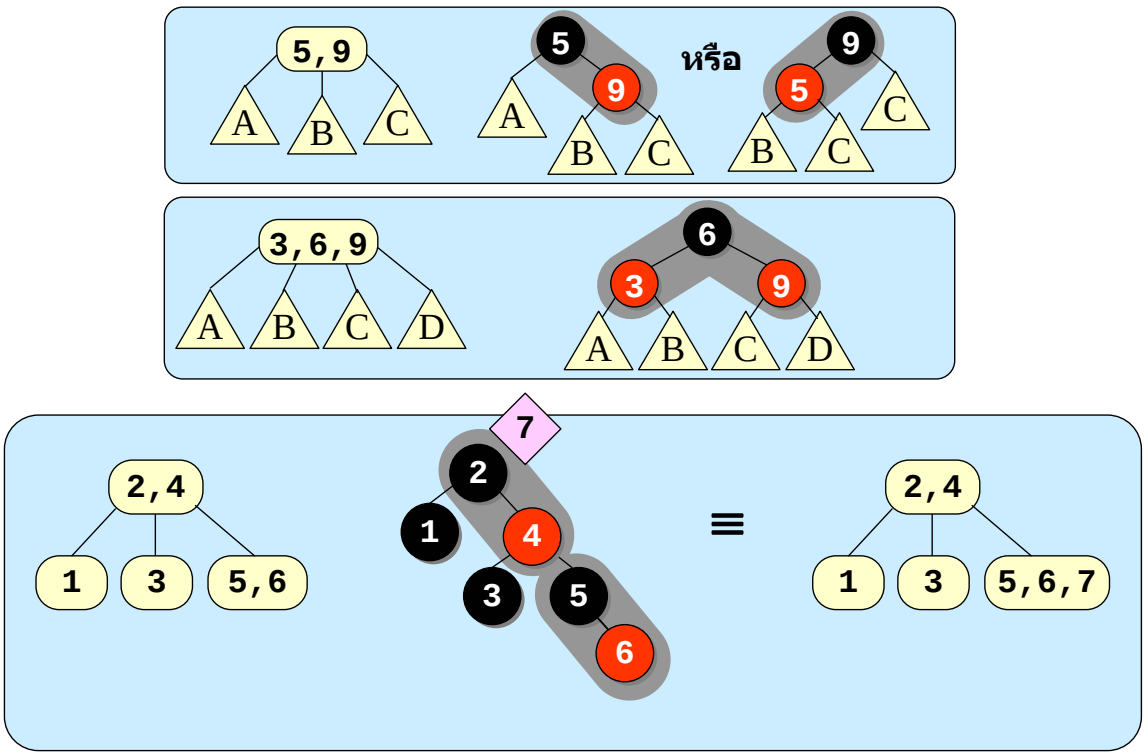
- มีสีกำกับปม, ปมแดงกับปมดำ
- เป็นรูปแบบการสร้างต้นไม้ 2-3-4 ด้วยปมแบบ 2
  - ปมแบบ 3 แทนด้วยปมแบบ 2 สองปม
  - ปมแบบ 4 แทนด้วยปมแบบ 2 สามปม



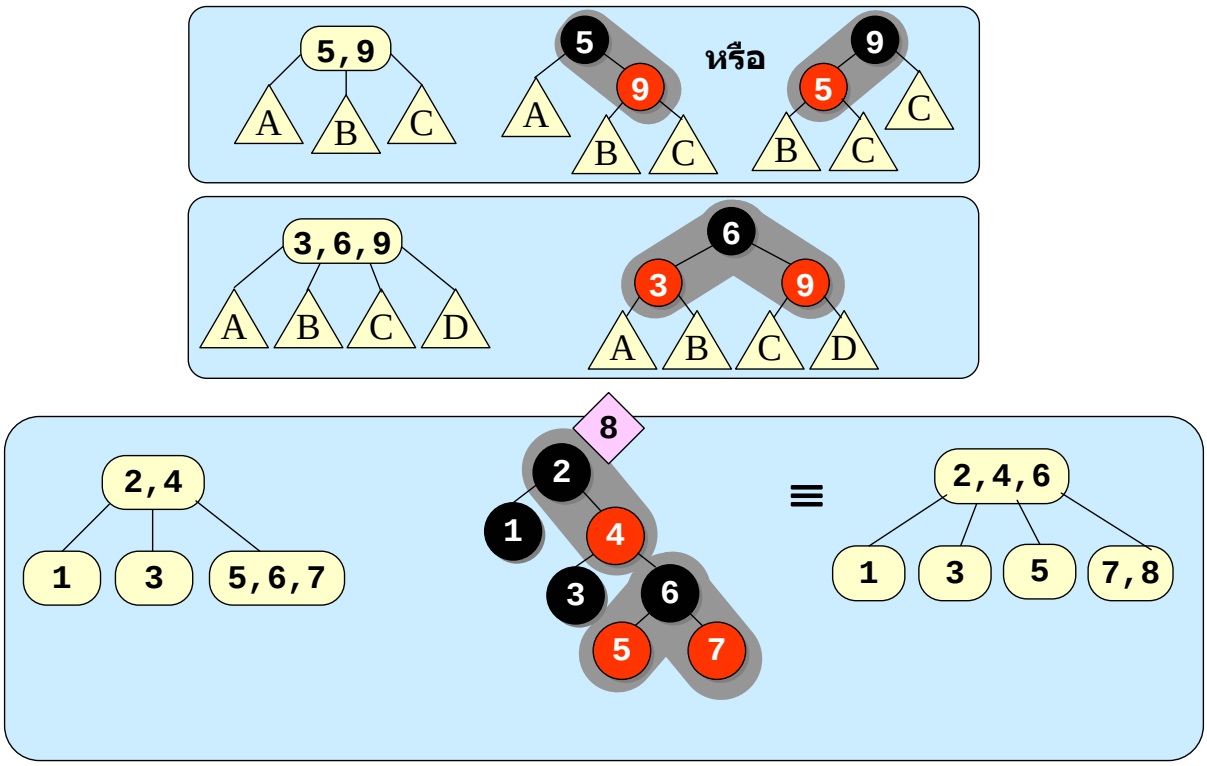
การเพิ่มข้อมูล



การเพิ่มข้อมูล



การเพิ่มข้อมูล



สรุป : ต้นไม้แดงดำ

- เป็นวิธีการสร้างต้นไม้ได้ดุล 2-3-4 แบบหนึ่ง
- แต่ละปมต้องใช้ 1 บิตเก็บสีกำกับปม
- ปมแบบ 3 และแบบ 4 เปลี่ยนเป็นแบบ 2
- ดังนั้นสูงอย่างมาก 2 เท่าของต้นไม้ได้ดุล 2-3-4
- เวลาในการค้น เพิ่ม ลบ เป็น O(log n)
- เป็นต้นไม้ค้นหาที่ใช้ในคลาสมาตรฐานของจาวา

สรุป

- ต้นไม้ค้นหามีอีกหลายแบบ
- แบบประกันความสูง
  - ต้นไม้เอวีแอล, ต้นไม้ได้ดุล 2-3-4, ต้นไม้แดงดำ
- แบบประกันประสิทธิภาพในกรณีเฉลี่ย
  - ต้นไม้ทรีพ
- แบบประกันประสิทธิภาพในกรณีถัวเฉลี่ย
  - ต้นไม้บาน