ค่ายอบรมโอลิมปิกวิชาการ 2

Data Structure: B-Tree

รัชดาพร คณาวงษ์

16 มีนาคม 2562

ศูนย์มหาวิทยาลัยศิลปากร

ทำไมต้องมีโครงสร้างต้นไม้แบบบี (b-tree)

- โครงสร้างการเรียงข้อมูลที่เราเรียกว่าอินเด็กซ์สำหรับชุดข้อมูลขนาดใหญ่บางครั้งไม่สามารถเก็บในหน่วยความจำหลักได้ ทำให้ต้องใช้วิธีการเก็บข้อมูลที่มีประสิทธิภาพมากขึ้น
- ถ้าจานแม่เหล็กหมุนได้ 3600 RPM, การเข้าถึงข้อมูลเกิดขึ้น 1/60 วินาทีหรือ 16.7 ms ถ้าเราใช้ AVL tree เพื่อเก็บข้อมูล 20 บ้านเรคคอร์ด ถึงแม้จะเป็นไบนารีทรีก็จะต้องมีความลึกมาก ทำให้เวลาในการเข้าถึงข้อมูลนานด้วย $\log_2 20{,}000{,}000 = 24$ หรือประมาณ 0.2 วินาที

ทำไมต้องมีโครงสร้างต้นไม้แบบบี (b-tree)

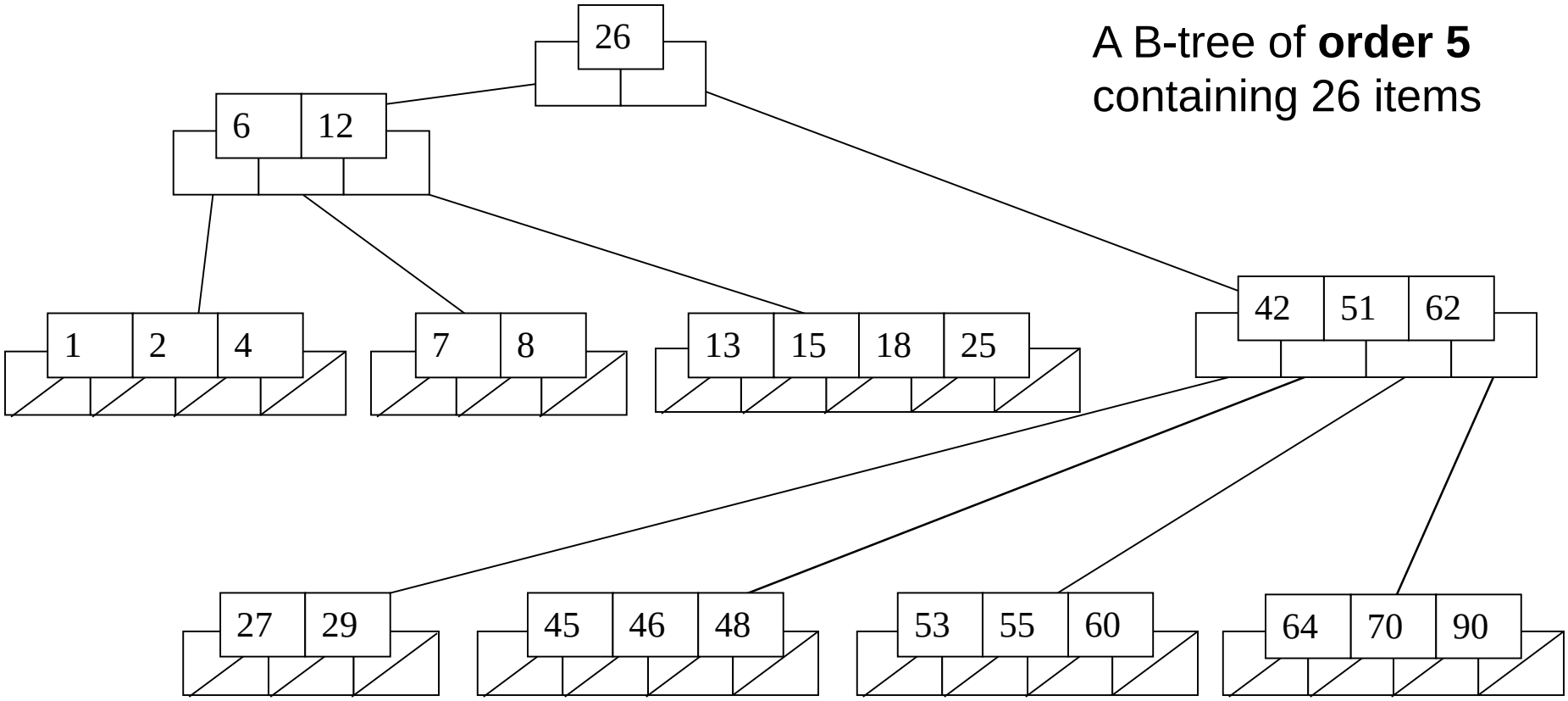
- เราไม่สามารถพัฒนาการค้นหาสำหรับไบนารีทรีให้เวลาในการค้นหาน้อยกว่า log2 n
- แต่เราสามารถเพิ่มกิ่งก้านทำให้ความสูงของต้นไม้ลดลง ก็จะทำให้การค้นหาเร็วขึ้นเพราะความสูงลดลง เส้นทางไปในแต่ละโหนดก็สั้นขึ้น

B-Tree

- เป็น Tree ที่มีคุณสมบัติแบบหลายทิศทาง ออกแบบมาเป็นพิเศษเพื่อใช้ในการเก็บข้อมูลในดิสก์ของเครื่องคอมพิวเตอร์
- **ทรีแบบหลายทิศทางตามจำนวนของ m (Multiway tree of order m)** หมายความว่า ในแต่ละ node จะมีเส้นที่เชื่อมโยงไปยัง node ลูกได้เท่ากับ m ดังนั้นแสดงว่าใน 1 node จะมี node ลูกได้ไม่มากกว่า m ข้อมูล และมีข้อกำหนดของ B-Tree ดังนี้
  1. จำนวนของ Key ในแต่ละ node ที่ไม่ใช่ node ใบจะมีจำนวนของ Key เท่ากับ *m* -1
  2. node ใบทั้งหมดจะอยู่ในระดับเดียวกัน
  3. node ทั้งหมดที่ไม่ใช่ node ใบยกเว้น node รากจะมี node ลูกได้น้อยที่สุด m/2 node
  4. node แม่ ในแต่ละ node ใบจะมี node ลูกได้ 1 ถึง m node
  5. node ใบจะมี Key ได้ไม่มากกว่า m – 1

ค่า m ควรเป็นจำนวนคี่

An example B-Tree



A B-tree of **order 5** containing 26 items

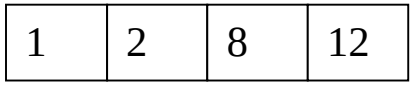
Note that all the leaves are at the same level

สร้าง B-tree

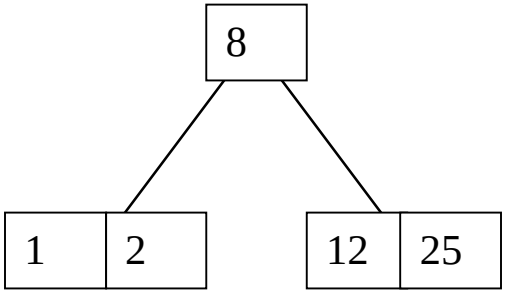
- สมมุด b-tree เป็นต้นไม้ว่าง และมีข้อมูลดังนี้:1 12 8 2 25 5 14 28 17 7 52 16 48 68 3 26 29 53 55 45
- ทำการสร้าง B-tree ที่มีลำดับ 5 นั้นหมายถึง 1 โหนดสามารถมีลิงค์สูงสุดได้ 5 ลิงค์และข้อมูลสูงสุดได้ 4 ข้อมูล
- ดังนั้นสีข้อมูลแรกจึงสามารถใส่ในโหนดรากได้

- ส่วนข้อมูลที่ 5 ไม่สามารถใส่ลงในโหนดรากได้อีกเพราะจะทำให้ไม่เป็นโครงสร้างของ B-tree order 5

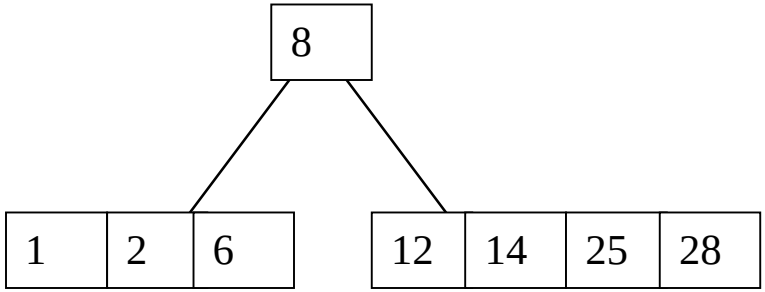
Constructing a B-tree (contd.)



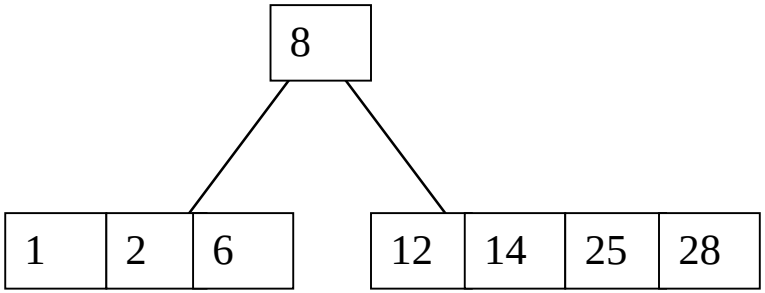
ดังนั้นเมื่อจะเพิ่ม 25 จะต้องเลือกข้อมูลกลางมาเป็นโหนดรากใหม่



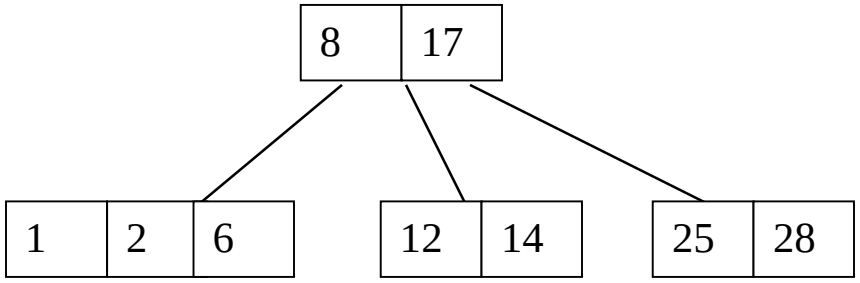
6, 14, 28 สามารถเพิ่มได้ที่โหนดใบ (leaf node) ได้



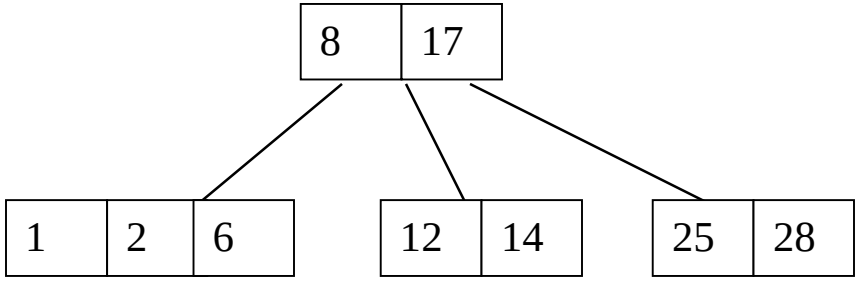
Constructing a B-tree (contd.)



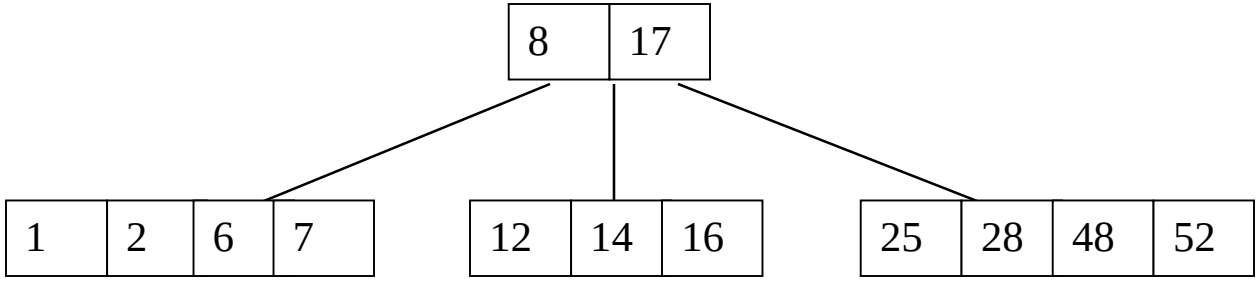
ถ้าเพิ่ม 17 ที่โหนดลูกขวาจะทำให้เข้อมูลเกิน ดังนั้นเราจึงต้องนำข้อมูลกลางมาไว้ทีรากแล้วทำการแบ่งข้อมูลออกมาเป็นอีกโหนด



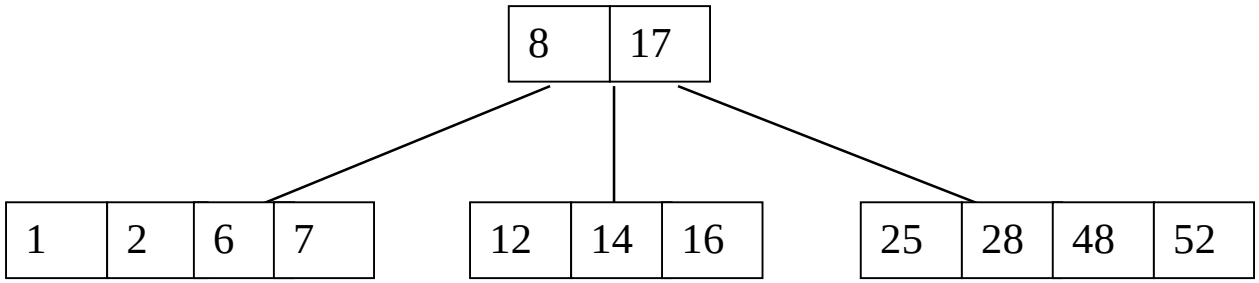
Constructing a B-tree (contd.)



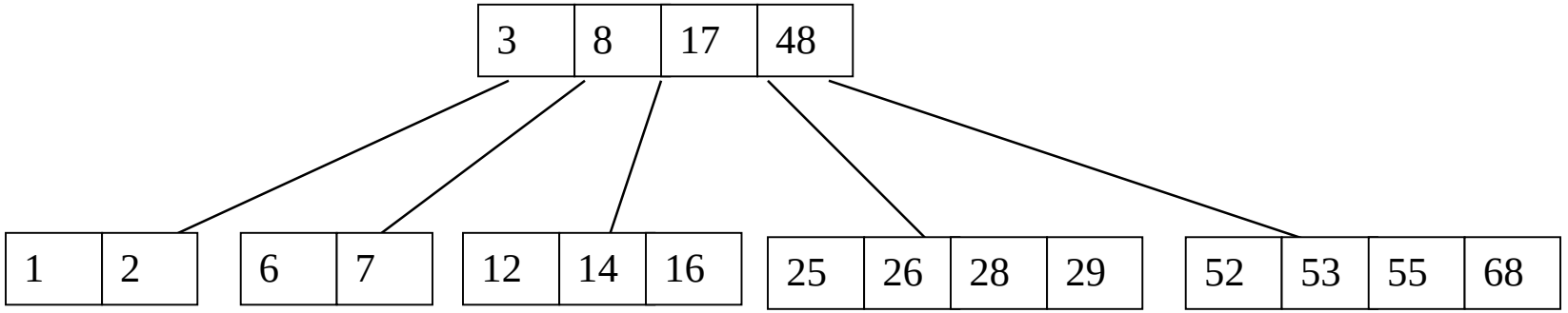
7, 52, 16, 48 เพิ่มที่โหนดใบ (leaf node)



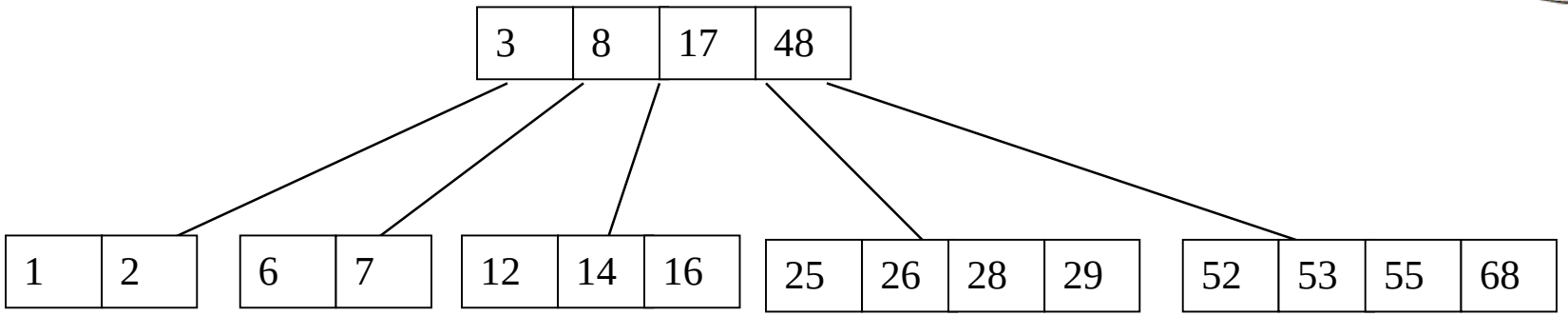
Constructing a B-tree (contd.)



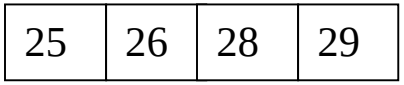
การเพิ่ม 68 ทำให้แยกโหนดใบขวาสุด แล้วย้าย 48 ไปที่ราก และเมื่อเพิ่ม 3 ทำให้โหนดใบซ้ายสุดแตกออกและให้ 3 ใส่ที่ราก ข้อมูล 26, 29, 53, 55 ใส่โหนดใบได้เลย



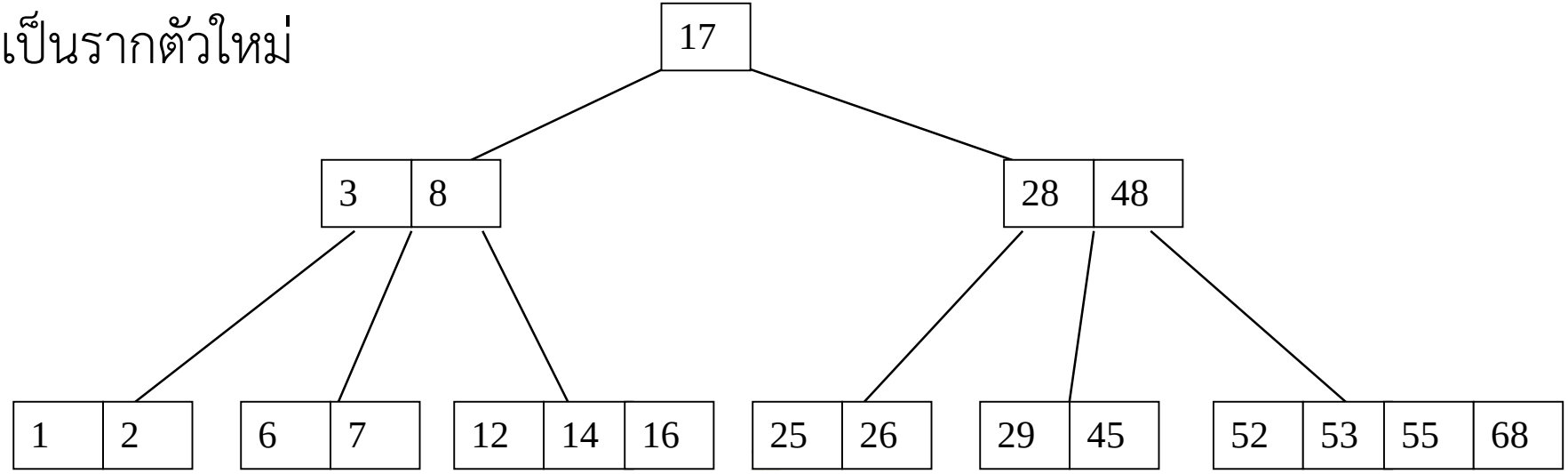
Constructing a B-tree (contd.)



เพิ่ม 45 ทำให้แยกโหนด



และทำให้ 28 เป็นข้อมูลรากซึ่งก็ทำให้รากต้องแตกออกและนำข้อมูลตรงกลางเป็นรากตัวใหม่



การเพิ่มข้อมูลใน B-Tree

- พยายามเพิ่มข้อมูลในโหนดใบก่อน
- ถ้าพบกว่าการเพิ่มข้อมูลในโหนดใบทำให้มีข้อมูลมากเกินไปให้แตกโหนดใบออกเป็นสองโหนด ย้ายข้อมูลตรงกลางไปที่โหนดพ่อแม่ของโหนดใบ
- ถ้าพบว่าโหนดพ่อแม่ไม่สามารถเก็บข้อมูลได้ให้ทำการแยกโหนดพ่อแม่ออกเป็นสองโหนดแล้วย้ายข้อมูลตรงกลางไปที่โหนดบรรพบุรุษของโหนดพ่อแม่
- วิธีนี้ให้ทำไปเรื่อยๆ จนถึงโหนดบนสุดที่สามารถใส่ข้อมูลได้
- ถ้าจำเป็นโหนดรากสามารถแตกออกเป็นสองโหนดและสร้างโหนดรากโหนดใหม่ด้วยข้อมูลตรงกลาง จะทำให้ต้นไม้มีความสูงเพิ่มขึ้นอีกหนึ่ง

Exercise in Inserting a B-Tree

ลองเพิ่มข้อมูลต่อไปนี้ใน 5-way B-tree:

3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 56

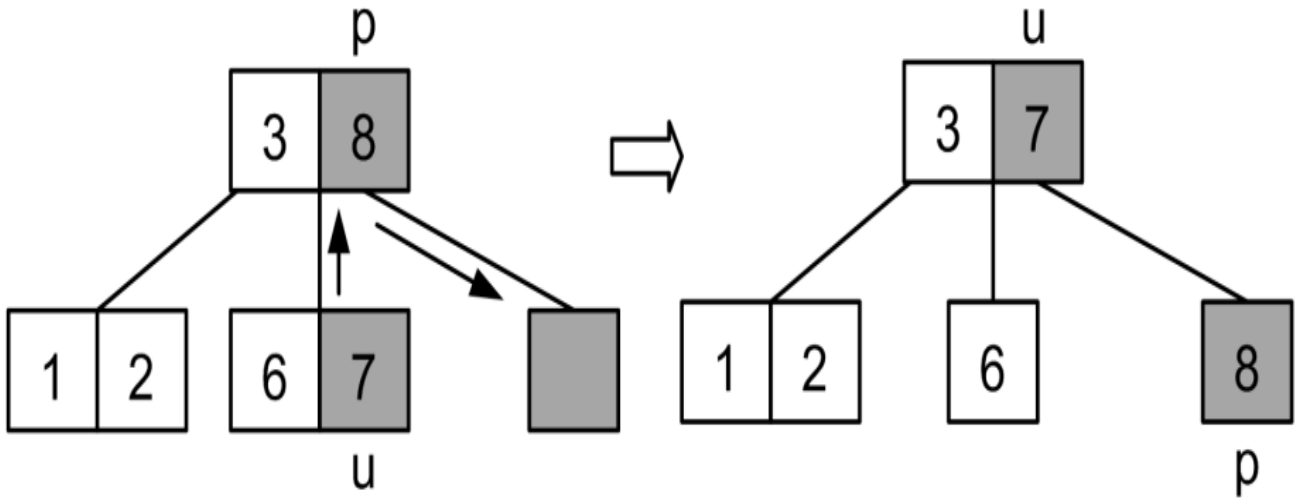
•

การลบ node ใน B-Tree

- การลบคีย์ใน B-Tree จะต้องลบข้อมูลในตำแหน่ง node ใบ
- ถ้าตำแหน่งคีย์ที่ต้องการลบอยู่ในตำแหน่งใบ และยังมีคีย์อื่นที่ไม่ใช้คีย์ที่ต้องการลบอยู่ใน node เดียวกัน ในกรณีนี้สามารถลบคีย์ออกจาก B-Tree ได้ทันที
- ถ้าตำแหน่งคีย์ที่ต้องการลบไม่ได้อยู่ในตำแหน่งใบ ให้ใช้หลักการ Inorder successor มาสลับตำแหน่งของคีย์ที่ต้องการลบกับคีย์ที่อยู่ในตำแหน่งใบ จึงลบคีย์ออกจาก B-Tree
- เมื่อลบคีย์แล้วส่งผลทำให้คีย์ที่อยู่ใน node มีจำนวนน้อยกว่าจำนวนของคีย์ที่กำหนดไว้ ให้พิจารณาดู node ในระดับพี่น้องดังต่อไปนี้

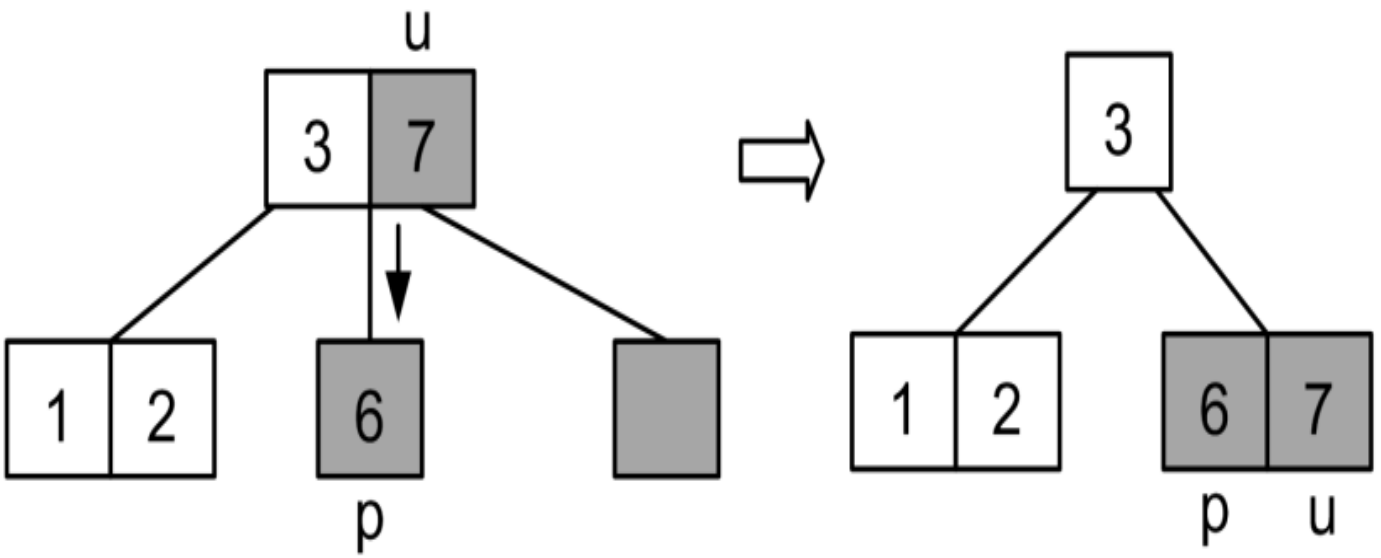
การลบ node ใน B-Tree

- ถ้า node ในระดับพี่น้องมีจำนวนของคีย์มากกกว่าหนึ่งคีย์ให้เลื่อนคีย์ในระดับพ่อแม่ลงมาแทนคีย์ที่ถูกลบไปและเลื่อนคีย์ในตำแหน่ง node พี่น้องขึ้นไปแทนคีย์ใน node พ่อแม่ที่ถูกเลื่อนลงไป ดังแสดงการเลื่อนตำแหน่งใน B-Tree



การลบ node ใน B-Tree

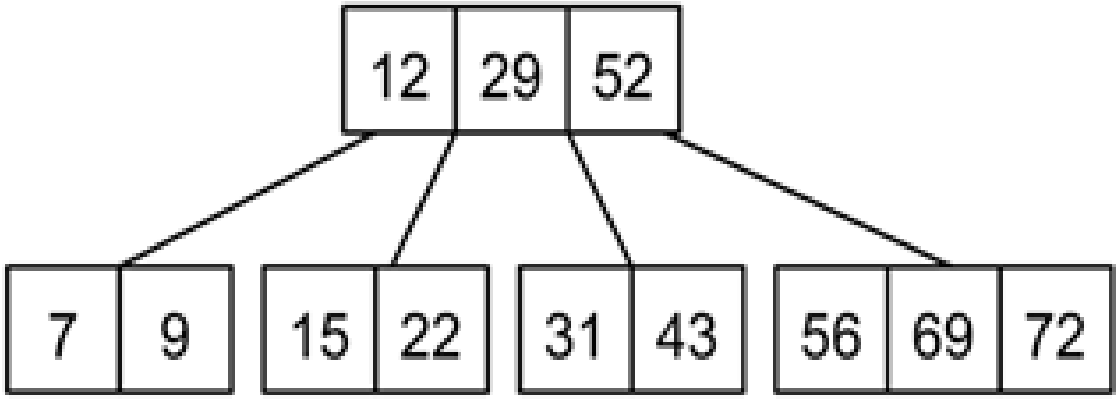
- ถ้า node ในระดับพี่น้องมีจำนวนของคีย์เท่ากับหนึ่ง ในกรณีนี้จะใช้หลักการรวมคีย์ โดยนำคีย์ในระดับพ่อแม่ลงไปรวมกับคีย์ในระดับพี่น้อง ดังแสดงการรวมคีย์ในรูป



การลบ node ใน B-Tree

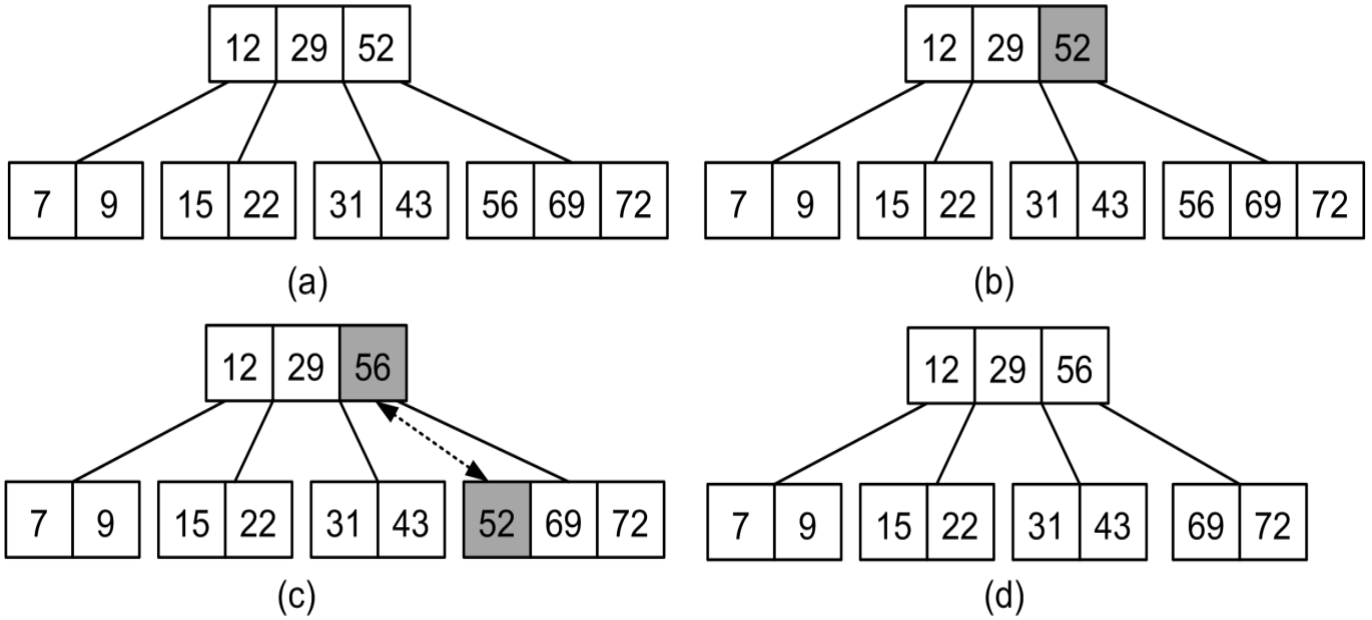
ตัวอย่าง การลบข้อมูลใน B-Tree

กำหนดให้มีลำดับการลบข้อมูลคือ 52, 72, 69, 56 ใน B-Tree โดยกำหนดให้ m = 5 แสดงได้ดังนี้



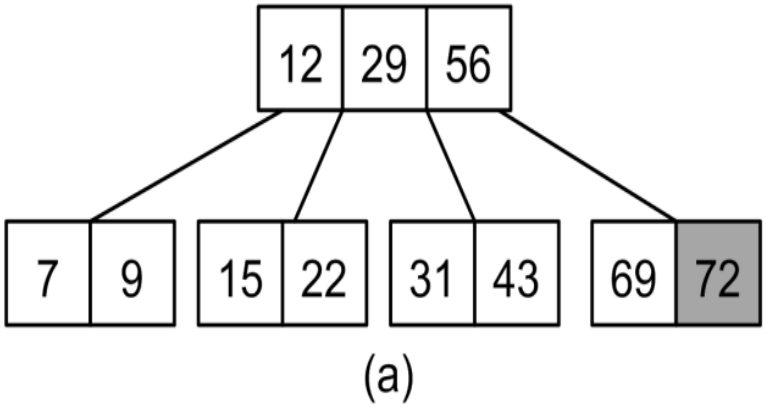
การลบ node ใน B-Tree

1. **ลบข้อมูล 52:** โดยนำ B-Tree ต้นแบบมาลบดังแสดงในรูป a เมื่อตรวจสอบข้อมูล 52 ที่ต้องการลบไม่ได้อยู่ในตำแหน่งใบต้องทำการสลับตำแหน่งข้อมูล 52 กับข้อมูลในตำแหน่งใบด้วยหลักการ Inorder successor คือข้อมูล 56 ดังแสดงในรูป (c) และทำการลบข้อมูล 52 ดังแสดงในรูป (d)

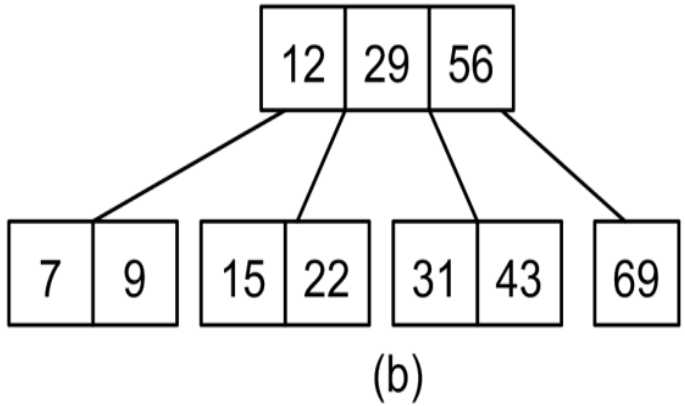


การลบ node ใน B-Tree

2. **ลบข้อมูล 72:** นำ B-Tree ที่เพิ่มข้อมูล 52 มาเป็นต้นแบบในการลบข้อมูล 72 เป็นข้อมูลที่อยู่ในตำแหน่งใบสามารถทำลบข้อมูล 72 ได้เลยดังแสดงในรูปที (b)



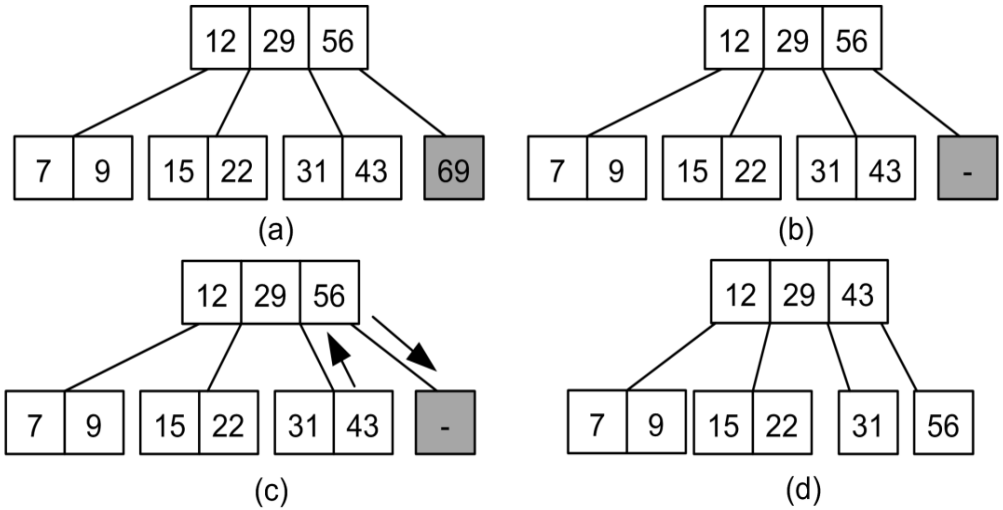
(a)



(b)

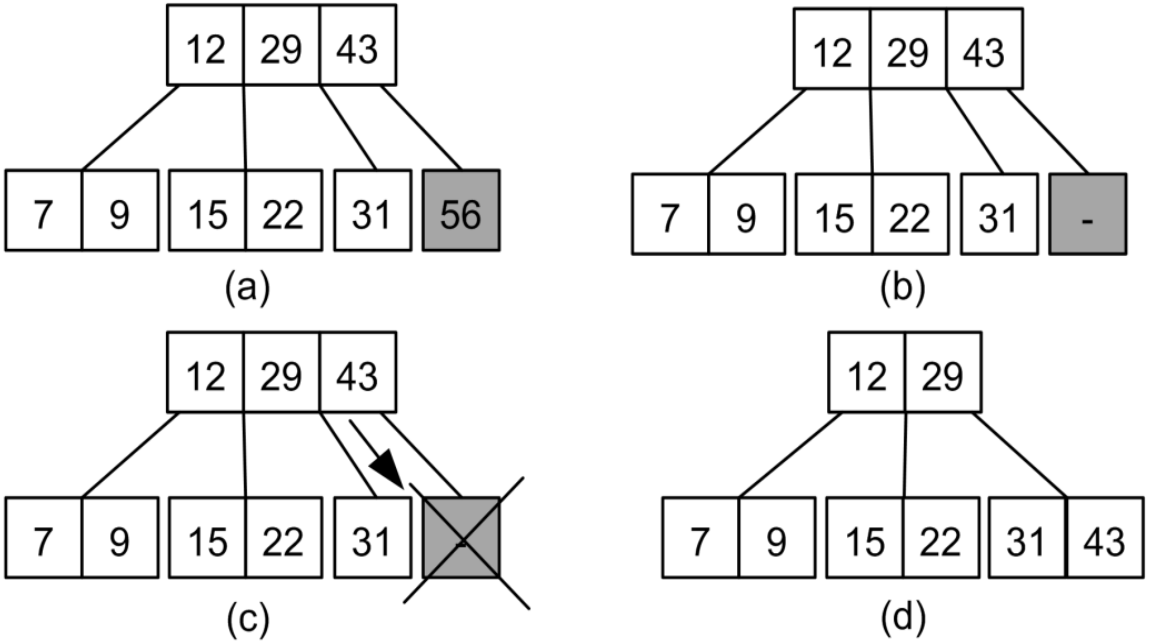
การลบ node ใน B-Tree

3. **ลบข้อมูล 69:** นำ B-Tree เพิ่มข้อมูล 72 มาเป็นทรีต้นแบบในการลบข้อมูล ข้อมูล 69 อยู่ในตำแหน่งใบสามารถทำการลบข้อมูล 69 ได้ทันทีแต่เมื่อทำการลบข้อมูล 69 แล้วข้อมูลแม่มีลูกอยู่เพียง node เดียวซึ่งไม่เป็นตามกฎของ B-Tree ดังนั้นจึงทำการเลื่อนข้อมูลในลำดับพี่สองคือ <31, 43> มี 2 ข้อมูลดังนั้นจึงทำการเลื่อนข้อมูล 56 ลงมาแทนข้อมูล 69 และทำการเลือก 43 ขึ้นไปเป็น node รากแทน ดังแสดงในรูป (d)



การลบ node ใน B-Tree

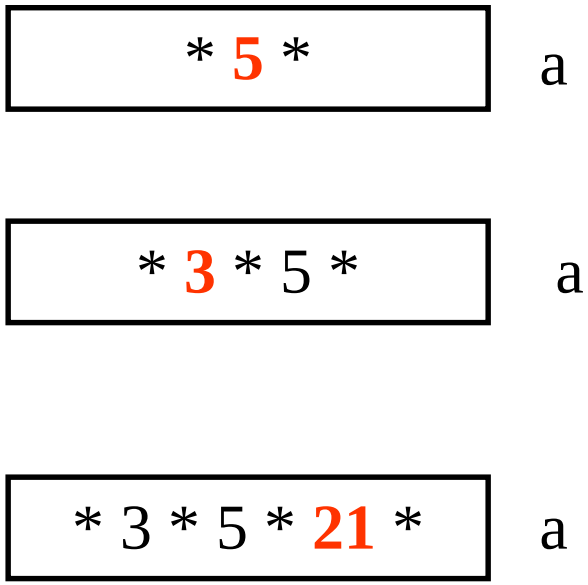
4. **ลบข้อมูล 56:** นำ B-Tree เพิ่มข้อมูล 69 มาเป็นต้นแบบ ข้อมูล 56 อยู่ในตำแหน่งใบสามารถลบข้อมูลได้ทันที แต่เมื่อลบไปแล้วแม่มีข้อมูลเพียงข้อมูลเดียวและเมื่อดู node ระดับพี่น้อง <31> มีเพียงข้อมูลเดียวดังนั้นต้องใช้หลักการเลื่อนข้อมูล 43 ไปร่วมกับ 31 ดังแสดงในรูปที่ (d)



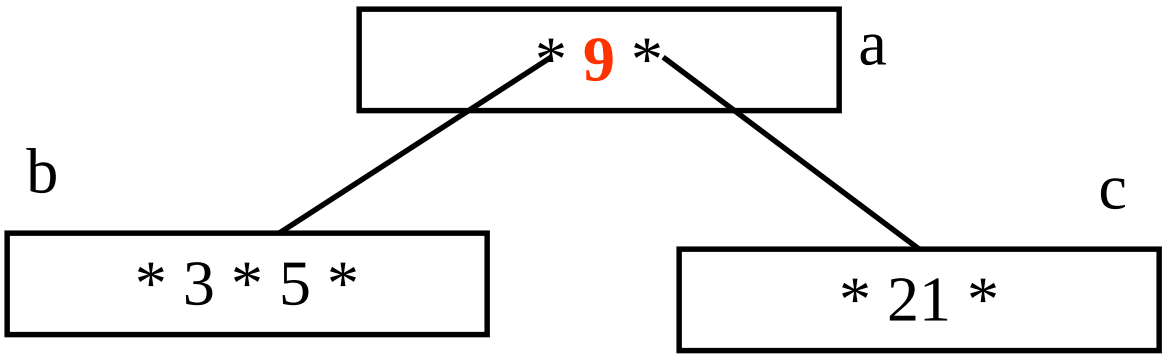
Example of B-Tree

- B-Tree of order 4
  - **แต่ละโหนดสามารถมีตัวชี้ได้สูงสุด 4 ตัว และ 3 ข้อมูล และอย่างน้อย 2 ตัวชี้ 1 ข้อมูล**
- Insert: 5, 3, 21, 9, 1, 13, 2, 7, 10, 12, 4, 8
- Delete: 2, 21, 10, 3, 4

Insert 5, 3, 21

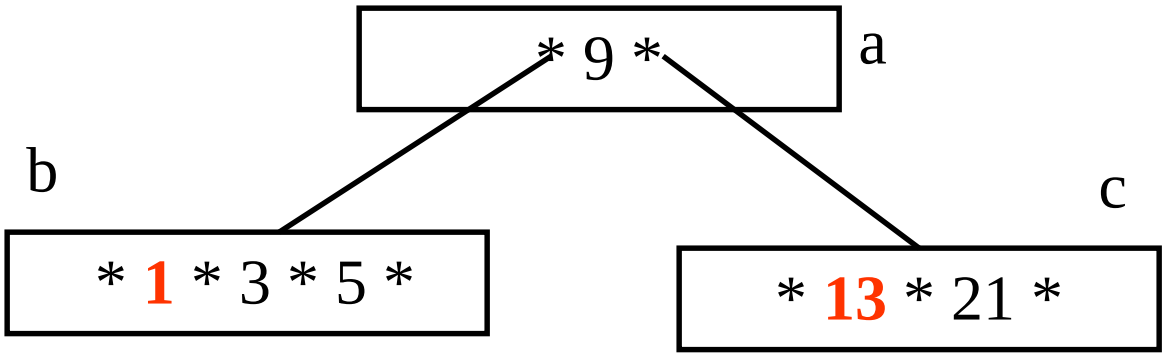


Insert 9



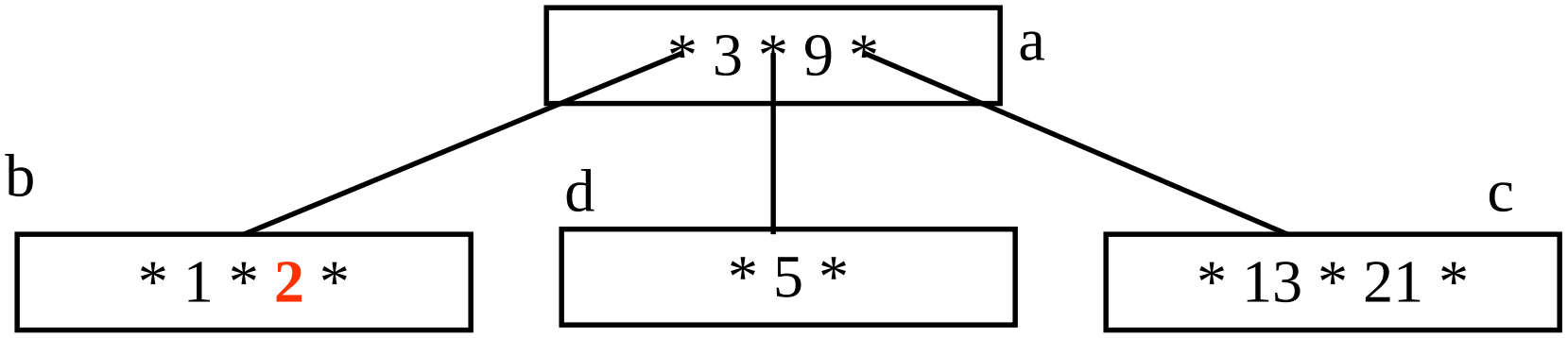
Node a splits creating 2 children: b and c

Insert 1, 13



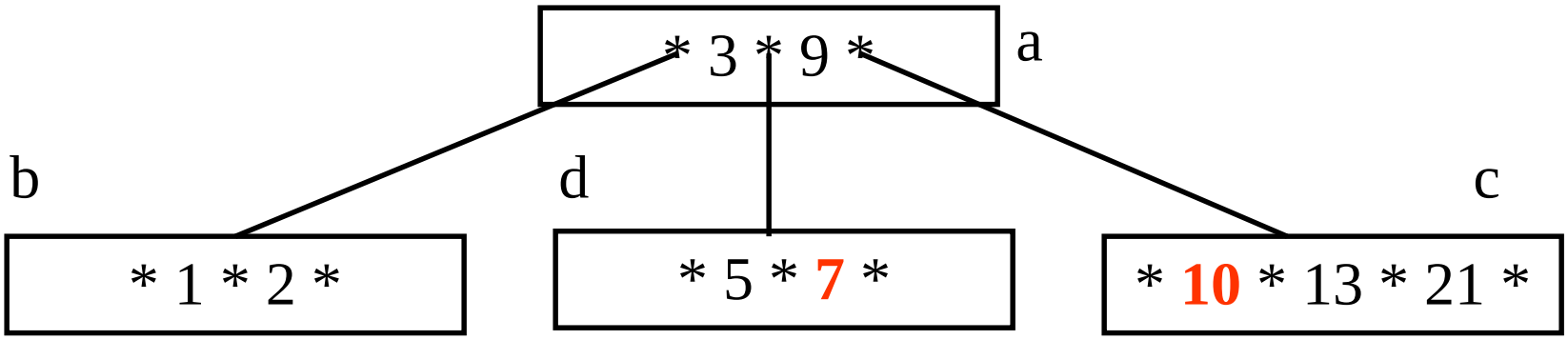
Nodes b and c have room to insert more elements

Insert 2



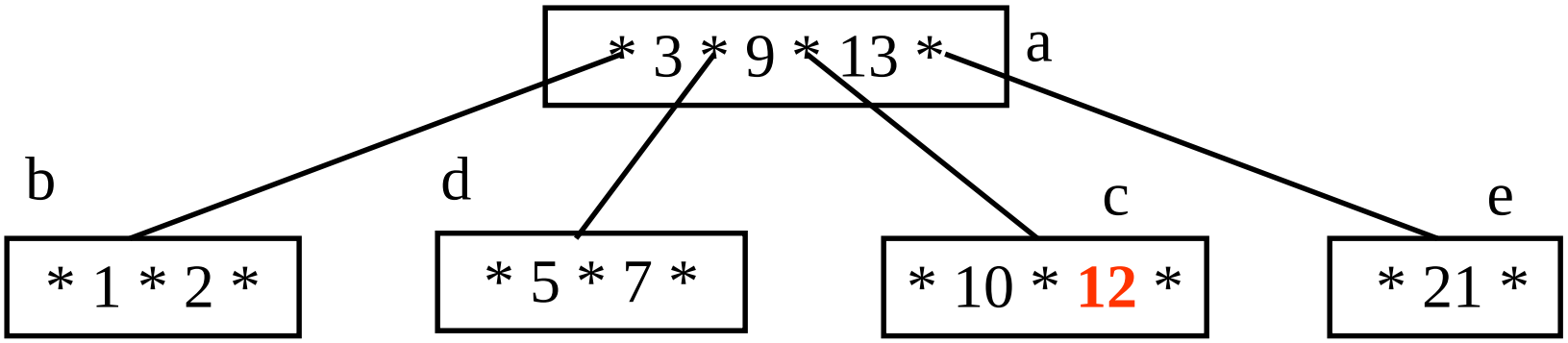
Node b has no more room, so it splits creating node d.

Insert 7, 10



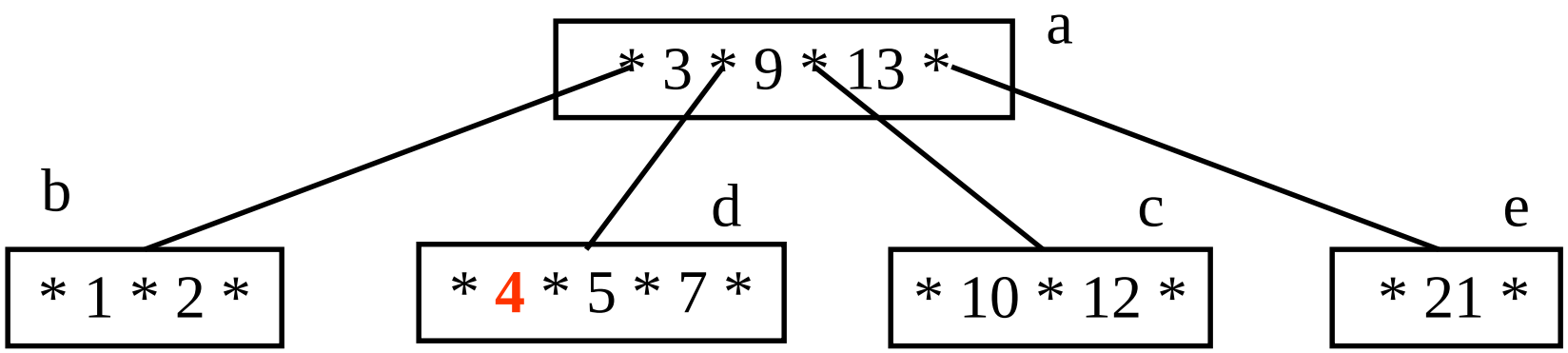
Nodes d and c have room to add more elements

Insert 12



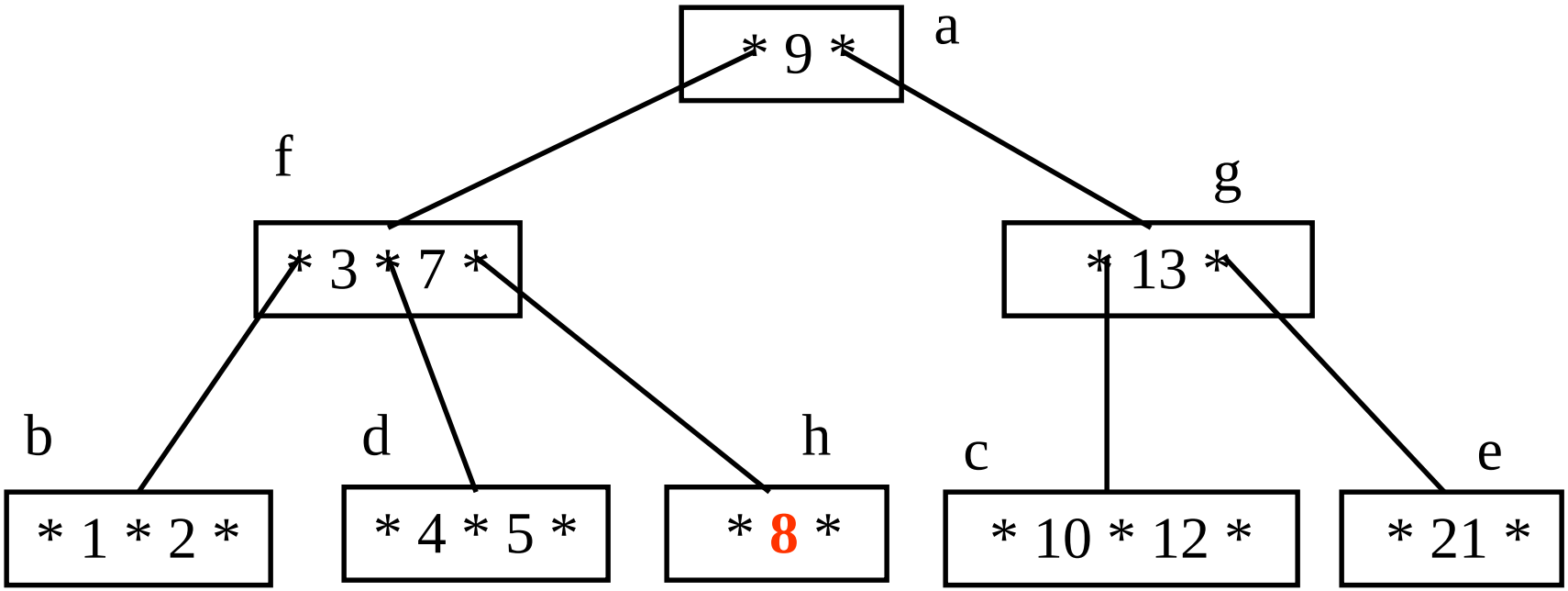
Nodes c must split into nodes c and e

Insert 4



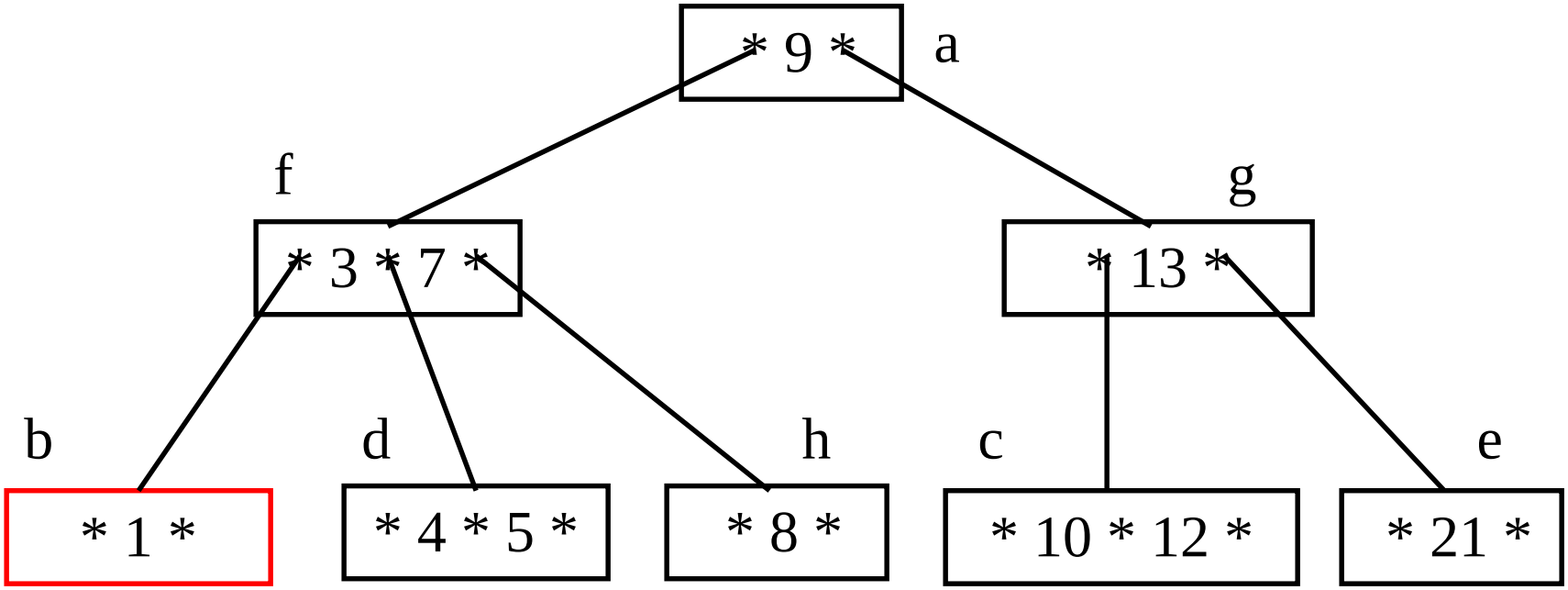
Node d has room for another element

Insert 8



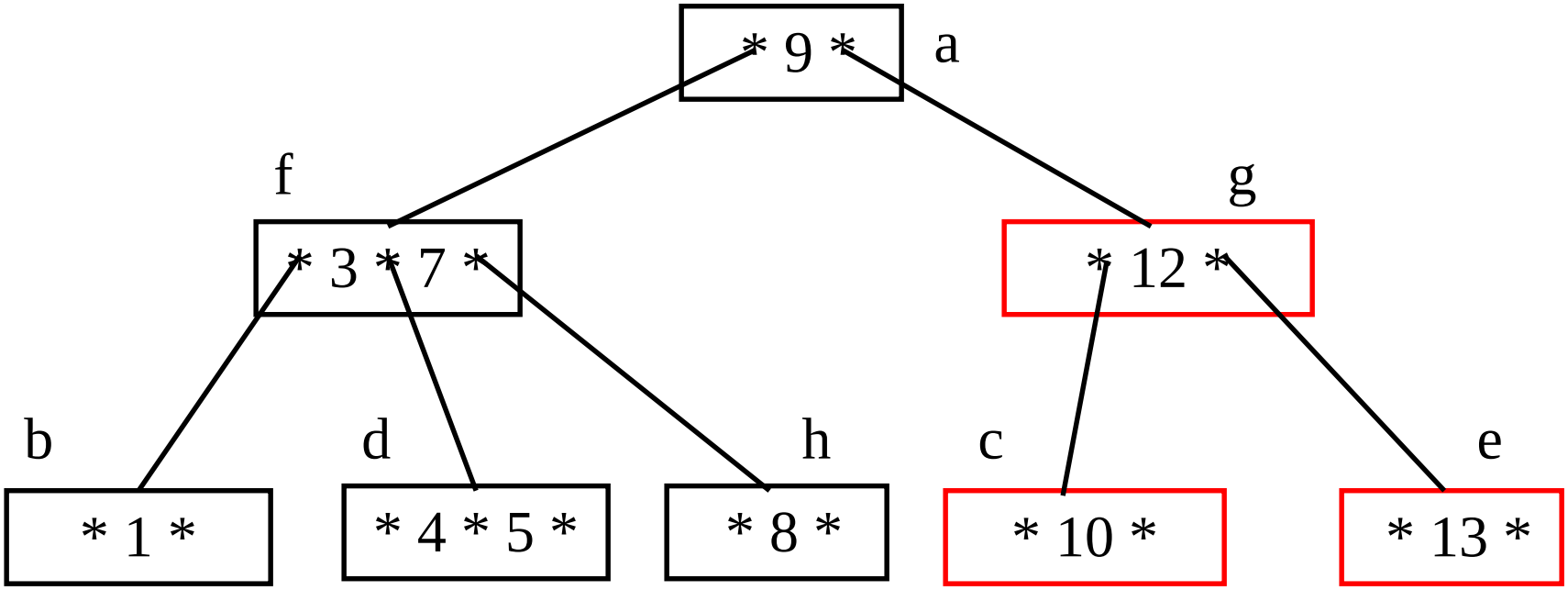
Node d must split into 2 nodes. This causes node a to split into 2 nodes and the tree grows a level.

Delete 2



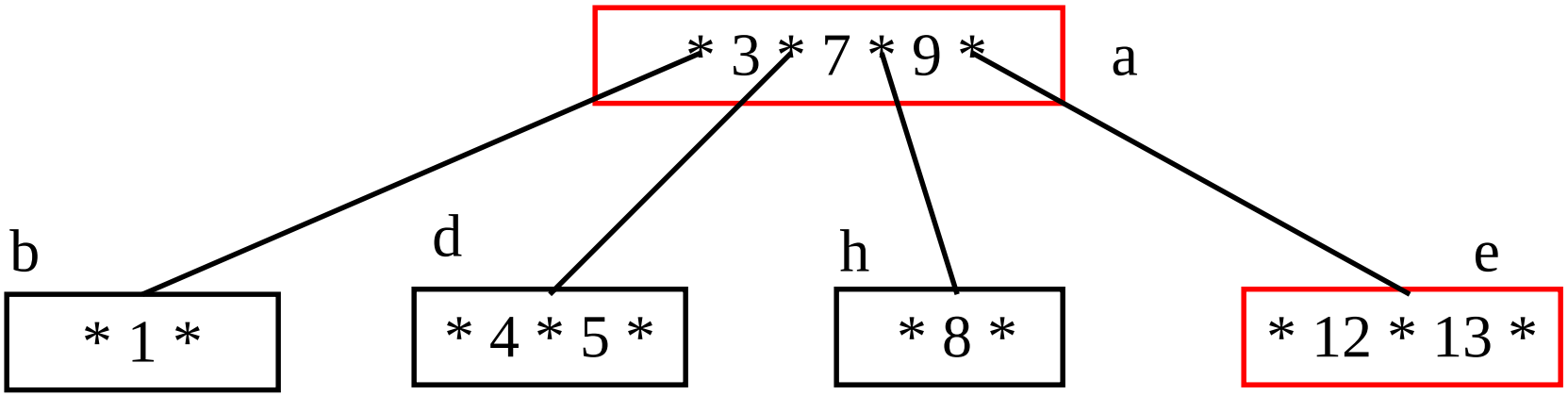
Node b can loose an element without underflow.

Delete 21



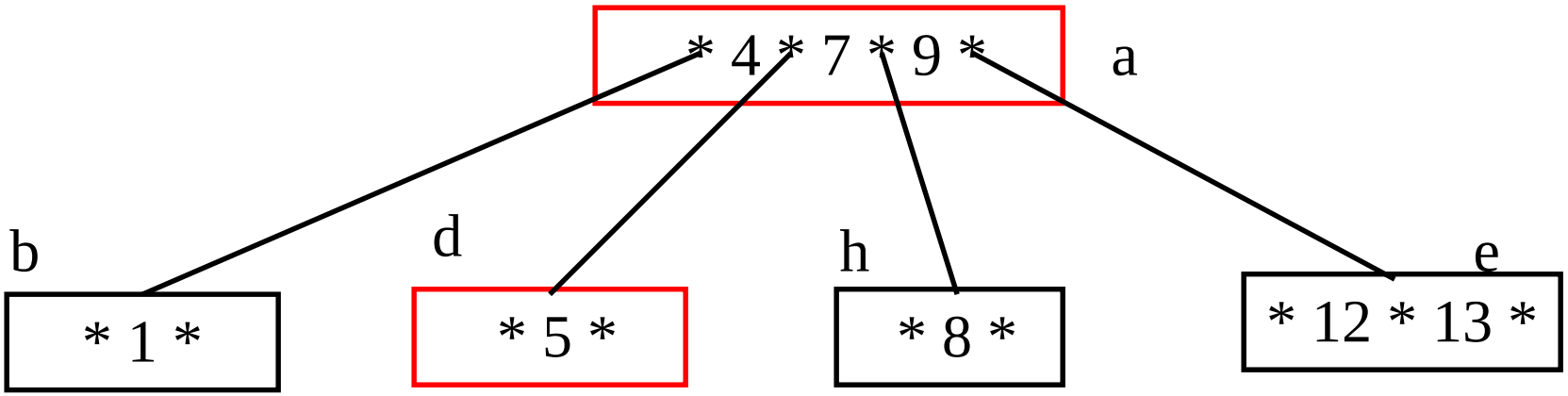
Deleting 21 causes node e to underflow, so elements are redistributed between nodes c, g, and e

Delete 10



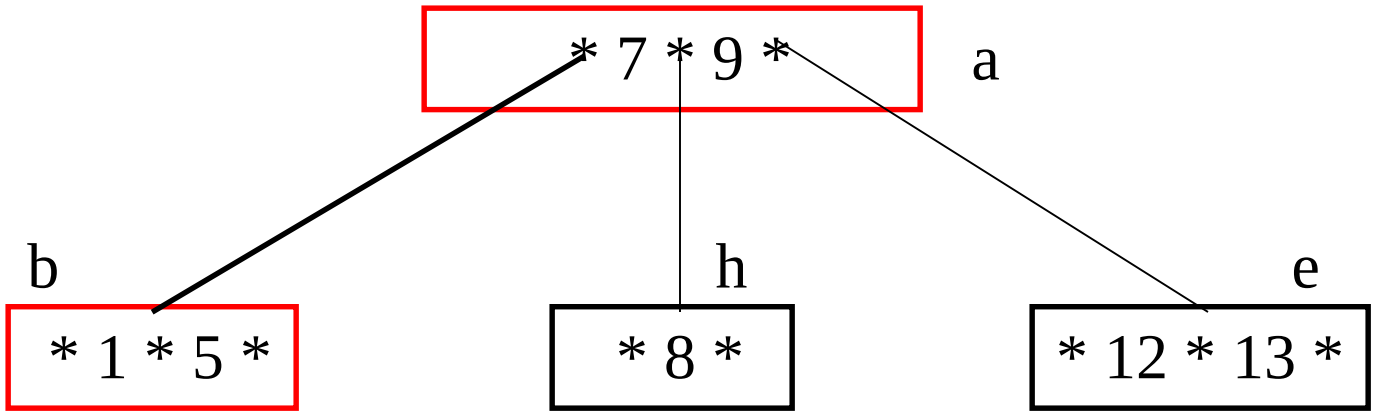
Deleting 10 causes node c to underflow. This causes the parent, node g to recombine with nodes f and a. This causes the tree to shrink one level.

Delete 3



Because 3 is a pointer to nodes below it, deleting 3 requires keys to be redistributed between nodes a and d.

Delete 4



Deleting 4 requires a redistribution of the keys in the subtrees of 4; however, nodes b and d do not have enough keys to redistribute without causing an underflow. Thus, nodes b and d must be combined.

Exercise

จงสร้าง 5-way B-tree โดยใช้ข้อมูลต่อไปนี้:

3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 56

และเพิ่มข้อมูลนี้: 2, 6,12

ลบข้อมูลต่อไปนี้: 4, 5, 7, 3, 14