Tree 2

โครงสร้างข้อมูลและอัลกอริทึมเบื้องต้น 305214 / 235012

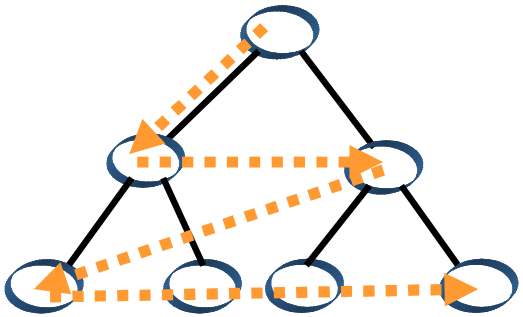
Topic

- Binary Tree Array Representation
- Expression Tree
- AVL Tree
- Huffman Tree

Binary Tree Array

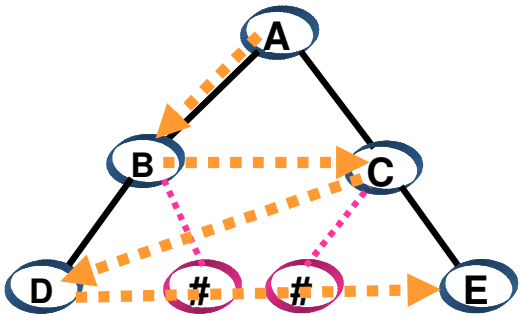
การใช้ array ในการเก็บโครงสร้างข้อมูลแบบ tree

- เริ่มต้นตำแหน่งแรกเก็บ root
- ตำแหน่งถัดไปเป็น level เดียวกันเรียงจากซ้ายไปขวา
- ตำแหน่งที่ว่างของ level เดียวกันให้ใส่ค่า dummy



Binary Tree Array

ตัวอย่าง



A	B	C	D	#	#	E

- เริ่มต้นจาก root
- ไล่ level ที่สองเรียงจากซ้ายไปขวา
- level ถัดไปทำเช่นเดียวกัน หากพบว่าว่างให้ใส่เป็นค่า dummy (ในที่นี้กำหนดเป็น #)

Binary Tree Array

จำนวนของ array ที่ต้องจอง

- มีการเพิ่มค่าเป็นผลบวกของ 2 ยกกำลังตาม level

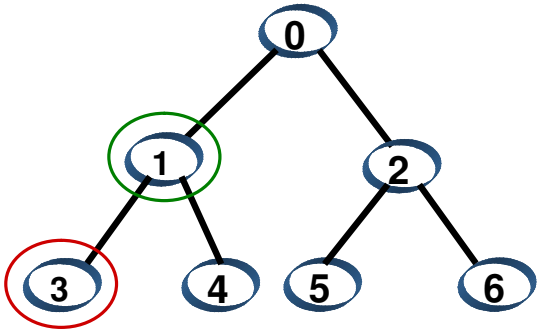
$2^0 + 2^1 + 2^2 + \ldots + 2^{n-1}$

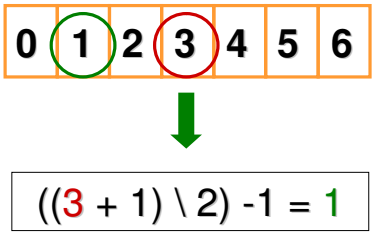
2 ชั้น → 3
3 ชั้น → 7
4 ชั้น → 15
n ชั้น → $2^n - 1$

Binary Tree Array

การหาตำแหน่งของโหนดพ่อของตำแหน่งที่ n

- กรณีเริ่ม array จาก 1 จะได้ พ่อเป็น n \ 2
- กรณีเริ่ม array จาก 0 จะได้พ่อเป็น ((n+1) \ 2) -1

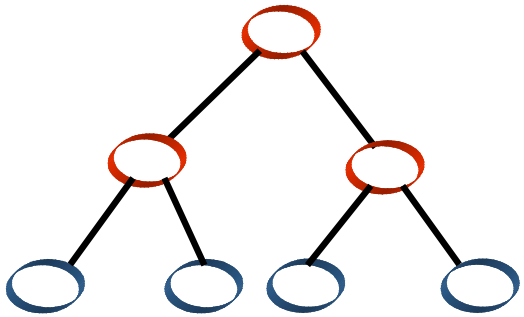


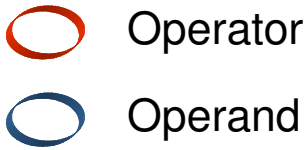


Expression Tree

การใช้ Binary Tree สำหรับเก็บนิพจน์การคำนวณ

- operand อยู่ที่ leaf node
- node อื่นๆ นอกจาก leaf ใช้เก็บ operator
- operator parent จะเป็นตัวกระทำกันระหว่าง child

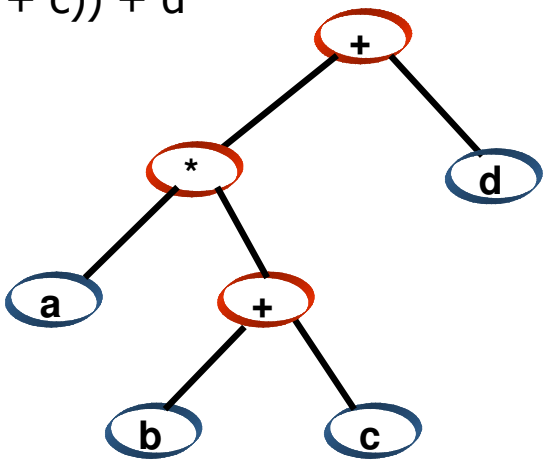




Expression Tree

ตัวอย่าง

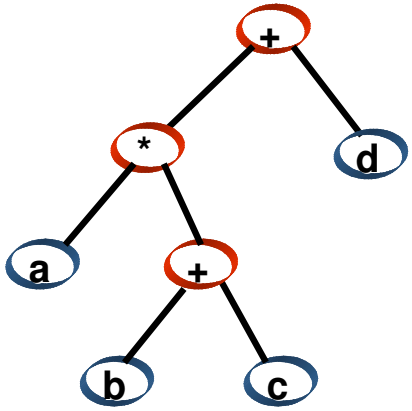
- (a * (b + c)) + d



Expression Tree

การท่อง Expression Tree

- ท่องแบบ Inorder ($T_L R T_R$) ได้นิพจน์ Infix

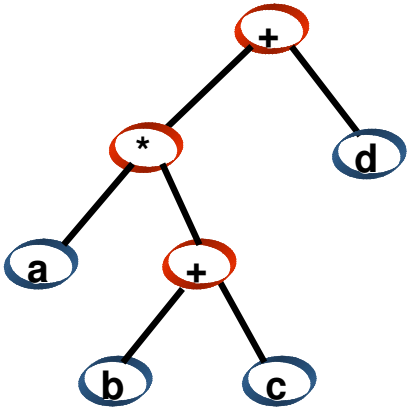


(a * (b + c)) + d

Expression Tree

การท่อง Expression Tree

- ท่องแบบ Preorder ($R T_L T_R$) ได้นิพจน์ Prefix

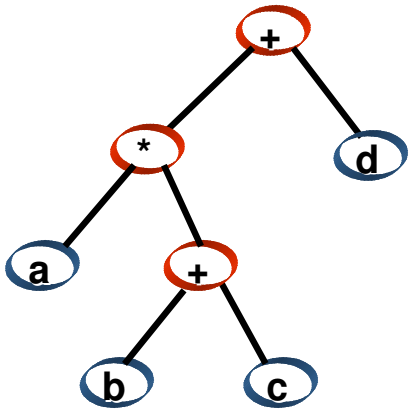


+ * a + b c d

Expression Tree

การท่อง Expression Tree

- ท่องแบบ Postorder ($T_L T_R R$) ได้นิพจน์ Prefix

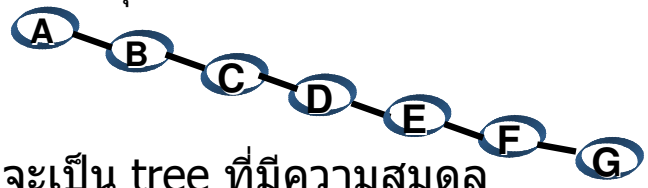


a b c + * d +

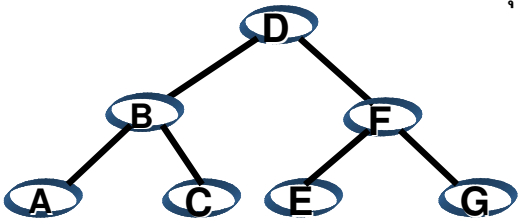
AVL Tree

เป็น BST ที่คำนึงถึงความสมดุลของ tree

- BST ปกติจะมีปัญหาเรื่องความสมดุลของ tree
- กรณีเลวร้ายที่สุดจะเป็น list



- กรณีที่ดีจะเป็น tree ที่มีความสมดุล



AVL Tree

พัฒนาโดยนักคณิตศาสตร์ชาวรัสเซีย

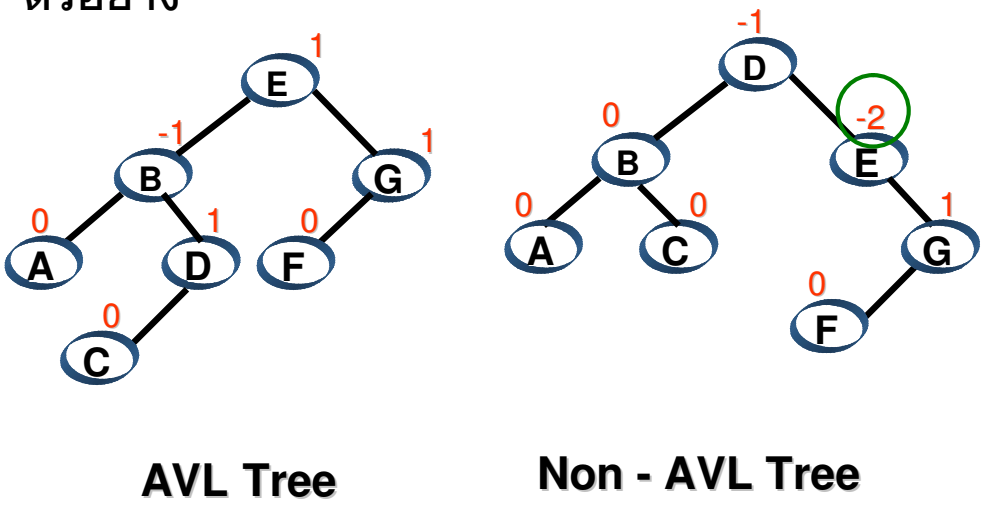
- G.M. Adelson-Velskii (**Georgy Maximovich**)
- E.M. Landis (**Yevgeniy Mikhailovich**)

มีการกำหนดค่า Balance Factor

- คำนวณจากความสูง sub tree ทางซ้าย – ความสูง sub tree ทางขวา
- ค่าที่ได้ต้องอยู่ระหว่าง -1 และ 1 (-1, 0, 1)

AVL Tree

ตัวอย่าง



AVL Tree **Non - AVL Tree**

AVL Tree

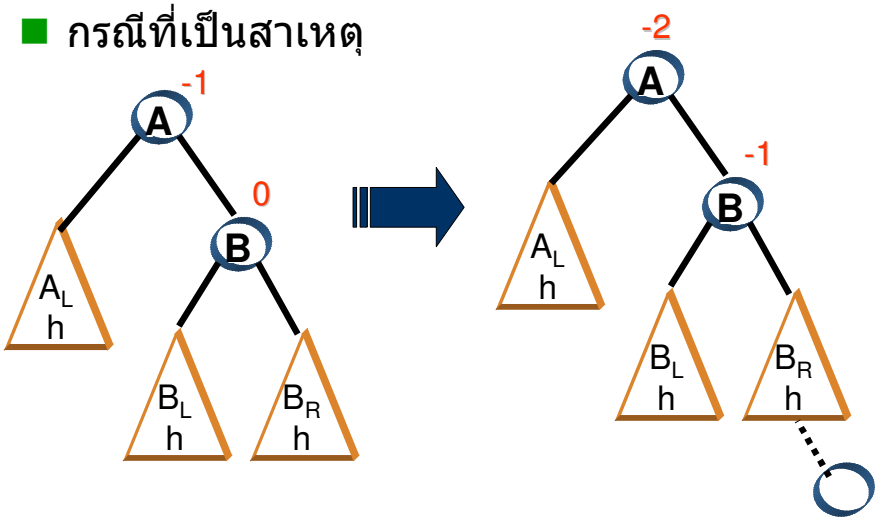
การสร้าง AVL tree

- ต้องสร้างตามเงื่อนไข Balance Factor (-1, 0, 1)
- เมื่อทำการเพิ่มโหนดลงใน BST ที่เป็น AVL อาจทำให้เกิดกรณีที่หลุดจากเงื่อนไข ต้องทำการแก้ไขโดยการทำ rotation
  - Single Left Rotation
  - Double Left Rotation
  - Single Right Rotation
  - Double Right Rotation

AVL Tree

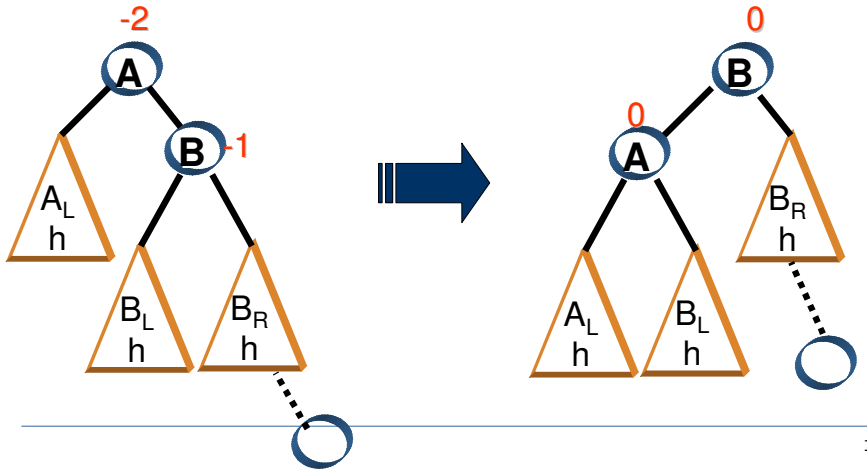
Single Left Rotation

- กรณีที่เป็นสาเหตุ



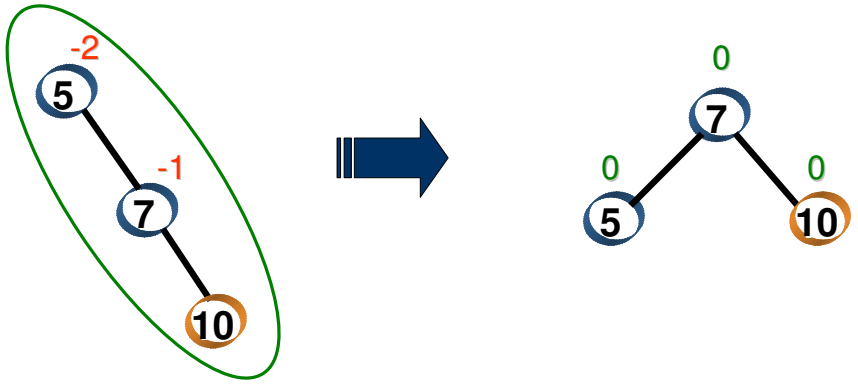
AVL Tree
Single Left Rotation

- การแก้ไข หมุนโหนด B ไปทางซ้ายแทน A แล้วต้นไม้ซ้ายของ B กลายเป็นต้นไม้ขวาของ A



AVL Tree
Single Left Rotation

- ตัวอย่าง

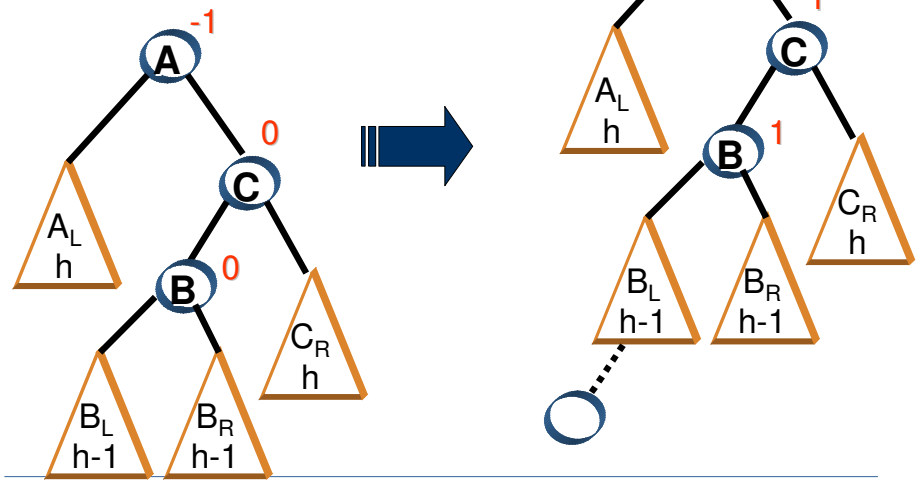


ค่าโหนดที่ผิดเงื่อนไขเป็น “ลบ”

ค่าโหนดลูกที่ทำให้โหนดผิดเงื่อนไขเป็น “ลบ”

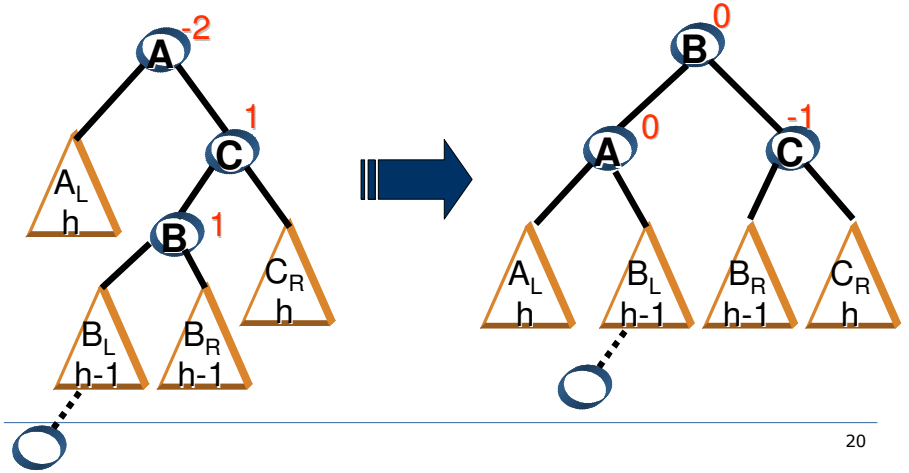
AVL Tree
Double Left Rotation

- กรณีที่เป็นสาเหตุ



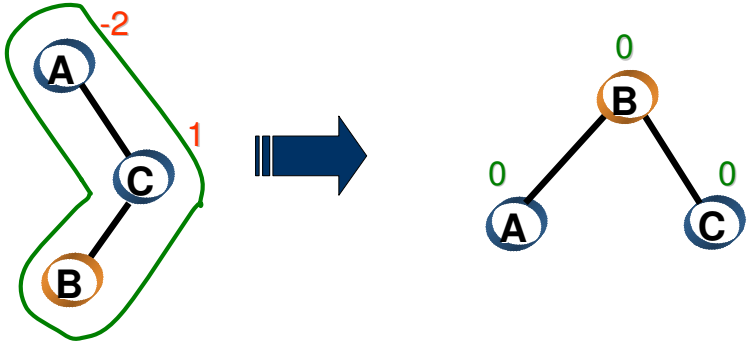
AVL Tree
Double Left Rotation

- การแก้ไข หมุนโหนด B ไปทางซ้าย 2 ครั้งแทน A แล้วต้นไม้ซ้ายของ B กลายเป็นต้นไม้ขวาของ A ขวา B เป็นซ้าย C



AVL Tree
Double Left Rotation

- ตัวอย่าง

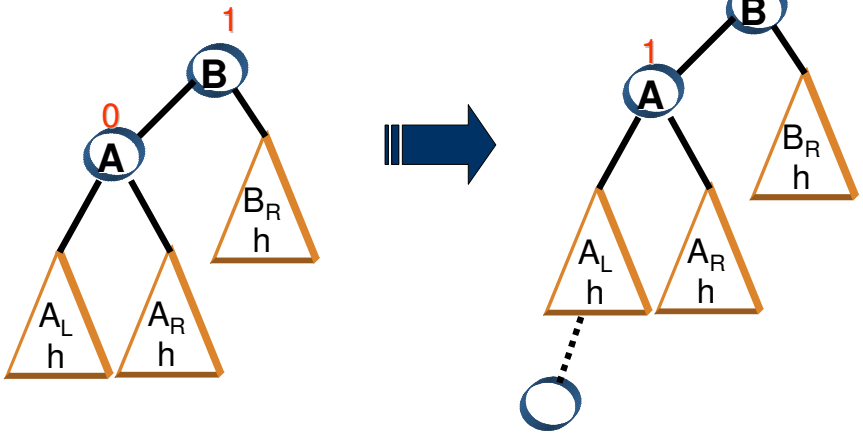


ค่าโหนดที่ผิดเงื่อนไขเป็น “ลบ”

ค่าโหนดลูกที่ทำให้โหนดผิดเงื่อนไขเป็น “บวก”

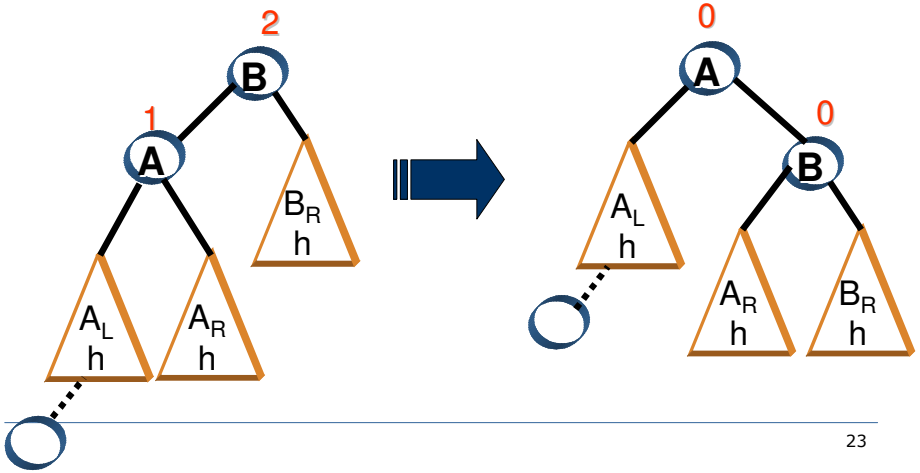
AVL Tree
Single Right Rotation

- กรณีที่เป็นสาเหตุ



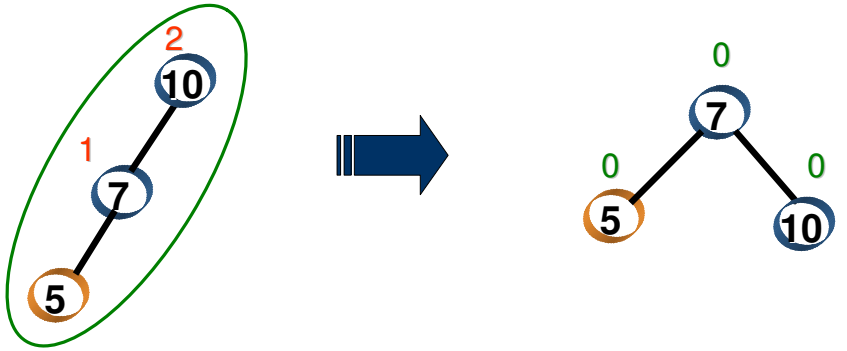
AVL Tree
Single Right Rotation

- การแก้ไข หมุนโหนด A ไปทางขวา 2 ของ B แล้วต้นไม้ขวาของ A กลายเป็นต้นไม้ซ้าย B



AVL Tree
Single Right Rotation

- ตัวอย่าง

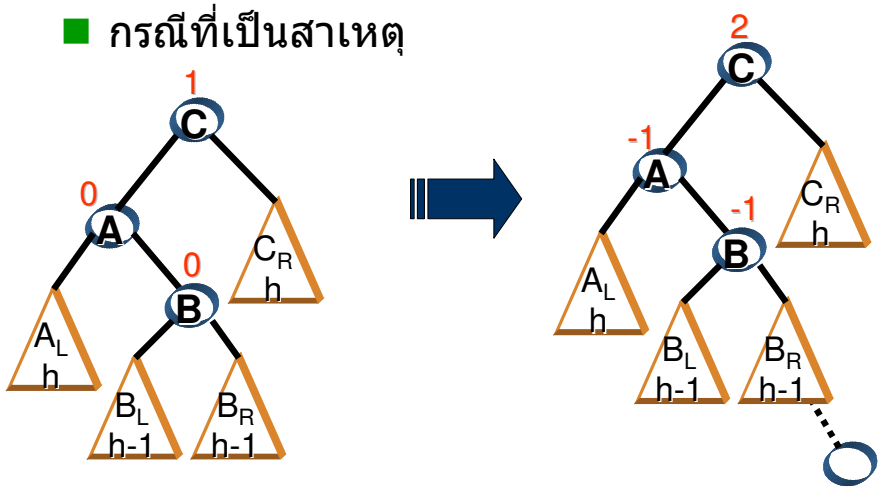


ค่าโหนดที่ผิดเงื่อนไขเป็น “บวก”

ค่าโหนดลูกที่ทำให้โหนดผิดเงื่อนไขเป็น “บวก”

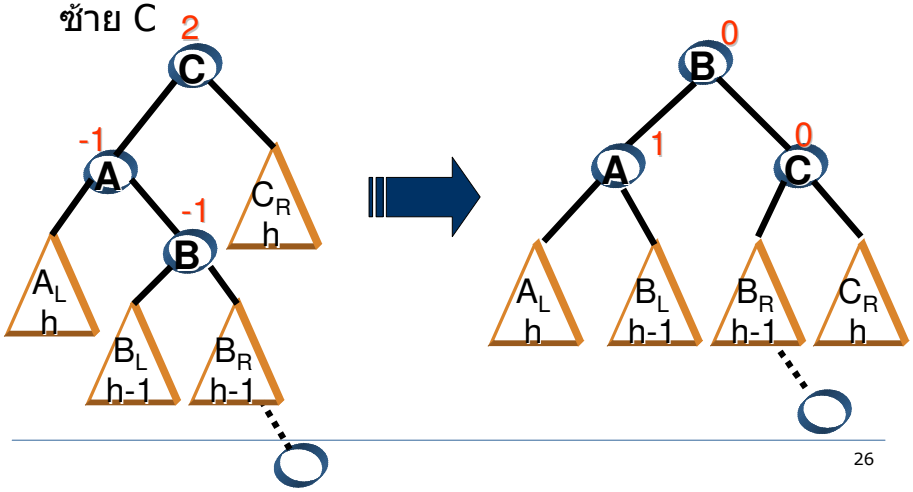
AVL Tree
Double Right Rotation

- กรณีที่เป็นสาเหตุ



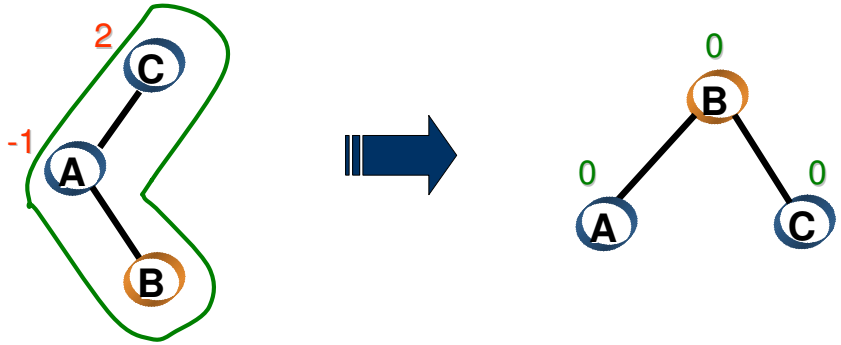
AVL Tree
Double Right Rotation

- การแก้ไข หมุนโหนด B ไปทางขวา 2 ครั้งแทน C แล้วต้นไม้ซ้ายของ B กลายเป็นต้นไม้ขวาของ A ขวา B เป็นซ้าย C



AVL Tree
Double Right Rotation

- ตัวอย่าง



ค่าโหนดที่ผิดเงื่อนไขเป็น “บวก”

ค่าโหนดลูกที่ทำให้โหนดผิดเงื่อนไขเป็น “ลบ”

Huffman Tree

เป็นการใช้ tree ในการเข้ารหัสแบบ Huffman

- การเข้ารหัสแบบ Huffman เป็นการเข้ารหัสแบบจำนวนบิทแบบแปรผัน

- มีเงื่อนไขในการเข้ารหัส 2 ประการ
  1. อักขระความถี่มากใช้จำนวนบิทน้อยกว่าความถี่น้อย
  2. อักขระใช้รหัสสั้นต้องไม่เป็นส่วนหนึ่งของอักขระรหัสยาว

Huffman Tree

ตัวอย่าง การเข้ารหัสแบบ Huffman

- กำหนดข้อมูล A B C A D B A E E B A

ได้ความถี่ของอักขระเป็น

อักขระ	ความถี่
A	4
B	3
C	1
D	1
E	2

Huffman Tree

ตัวอย่าง การเข้ารหัสแบบ Huffman

นำมาสร้างคิวโดยเรียงลำดับจากความถี่น้อยไปมาก หากความถี่เท่ากันให้เรียงตามตัวอักษร

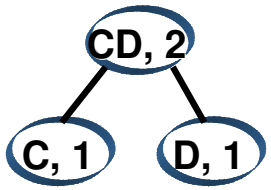
C, 1	D, 1	E, 2	B, 3	A, 4	

*หมายเหตุ คิวที่ใช้เป็นคิวแบบเรียงลำดับตามความถี่ การ enqueue จะใส่เข้าไปในตำแหน่งที่เรียงลำดับ

Huffman Tree

ตัวอย่าง การเข้ารหัสแบบ Huffman

ถ้าคิวไม่ว่างให้ dequeue ออกมา 2 โหนดมาสร้างต้นไม้ โดยให้โหนดพ่อเป็นผลรวมของ 2 โหนดที่ dequeue ออกมา



E, 2	B, 3	A, 4	

ถ้าคิวไม่ว่างให้ enqueue โหนดพ่อเข้าไปใหม่

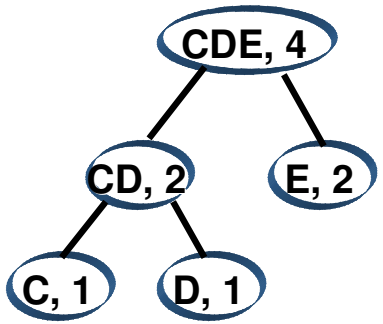
CD, 2	E, 2	B, 3	A, 4	

Huffman Tree

ตัวอย่าง การเข้ารหัสแบบ Huffman

ทำวนซ้ำไปเรื่อยๆ จนกว่าจะ dequeue หมด

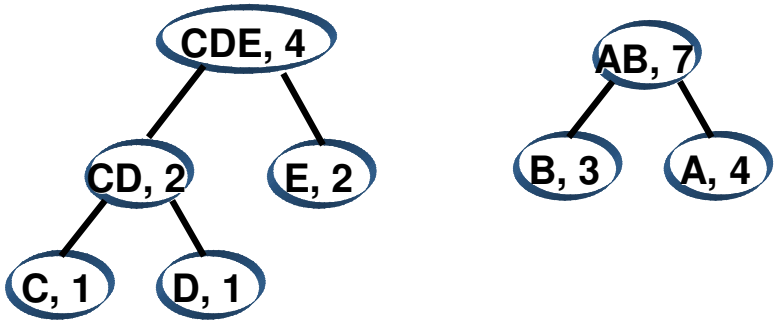
CD, 2	E, 2	B, 3	A, 4	



Huffman Tree

ตัวอย่าง การเข้ารหัสแบบ huffman

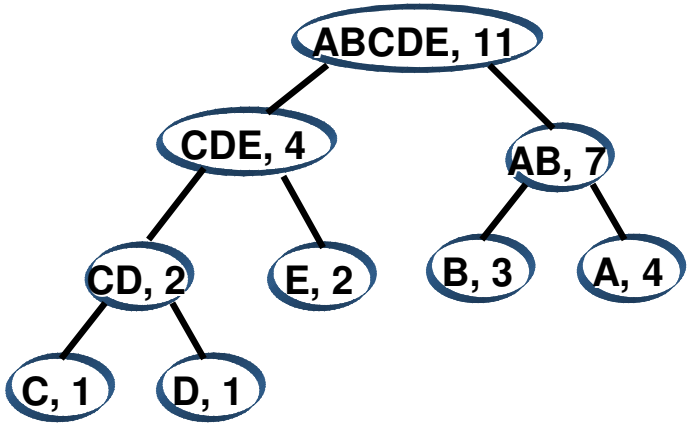
B, 3	A, 4	CDE, 4	



Huffman Tree

ตัวอย่าง การเข้ารหัสแบบ huffman

CDE, 4	AB, 7	



Huffman Tree

การเข้ารหัสแบบ huffman

- การเข้ารหัสเริ่มจาก root วิ่งทางซ้ายเป็น 0 วิ่งทางขวาเป็น 1

จากต้นไม้ได้รหัสเป็น

อักขระ	รหัส
A	11
B	10
C	000
D	001
E	01

Huffman Tree

การถอดรหัสแบบ Huffman

- เริ่มจาก root bit 0 วิ่งซ้าย bit 1 วิ่งขวา ถึง leaf หมดหนึ่งอักขระ

ตัวอย่าง 111000011

ABCA