

CANTHO UNIVERSITY

Chapter 5: AVL Tree (GM **Adelson - Velsky** & EM **Landis**)

Lâm Hoài Bảo - FSE – CICT
Trương Minh Thái – FSE - CICT

www.ctu.edu.vn

1




CANTHO UNIVERSITY

Agenda

- Introduction
- Rotations
- Operators
- Summary

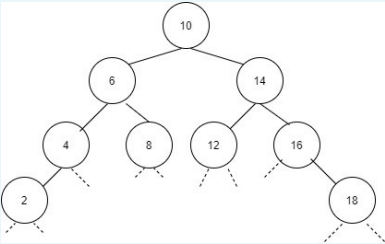
www.ctu.edu.vn

2

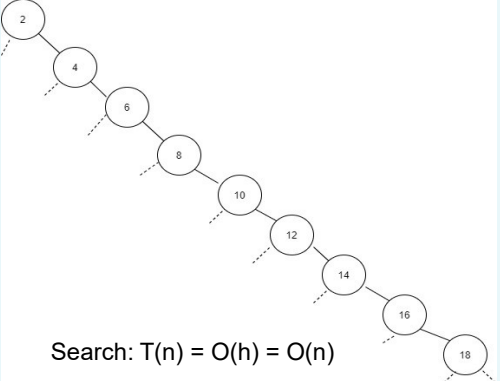


Balanced problem

- Given a collection of keys: 2, 4, 6, 8, 10, 12, 14, 16, 18
- Some BSTs




Search: $T(n) = O(h) = O(\log n)$



Search: $T(n) = O(h) = O(n)$

www.ctu.edu.vn
3

3



Balanced binary search tree

- A BST is balanced if $h = O(\log n)$ (h : the height of the tree).
- Goal: Maintain the BST balanced in dynamic operators (insert, delete)
- Some balanced BSTs
 - AVL trees
 - 2-3 trees
 - B trees
 - Red Black trees
 - Splay trees
 - ...

www.ctu.edu.vn
4

4

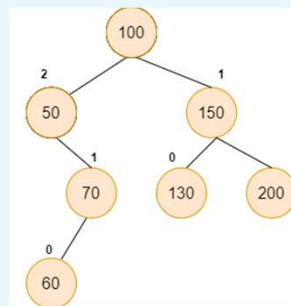


AVL tree

- A **BST** where the difference between the height of left child & right child of any node does not exceed 1.
- **Balanced factor**:
 - $BF(p) = \text{height}(\text{left}(p)) - \text{height}(\text{right}(p))$
- **Recall**: The height of a node is the length of the path from it to the farthest descendant leaf node.
 - A **NULL** node has the length -1
 - A **leaf** node has the length 0
 - $\text{height}(p) = 1 + \text{MAX}(\text{height}(\text{left}(p)), \text{height}(\text{right}(p)))$
- A node is called height-balanced if its balanced factor is either -1, 0, or 1



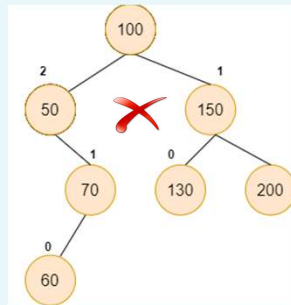
Examples





CANTHO UNIVERSITY

Examples



www.ctu.edu.vn

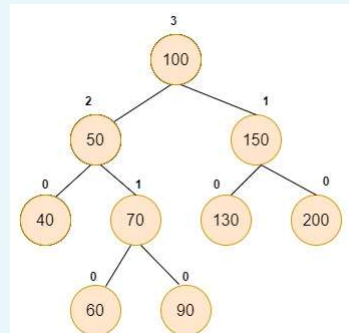
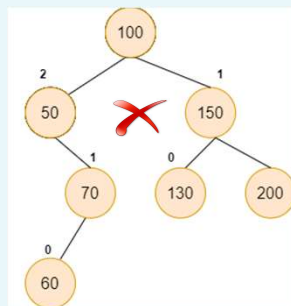
7

7



CANTHO UNIVERSITY

Examples



www.ctu.edu.vn

8

8

Examples


www.ctu.edu.vn 9

9

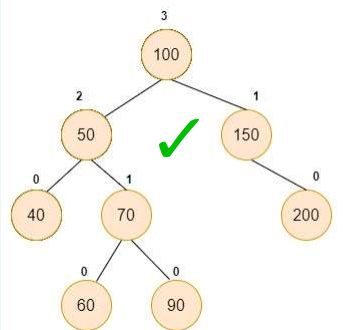
Examples

www.ctu.edu.vn 10

10




Examples



www.ctu.edu.vn
11

11



The height of an AVL tree

- Let $N(h)$ be the number of nodes of a tree which has the height h .
- $N(h)$
 - $N(-1) = 0$
 - $N(0) = 1$
 - $N(h) \geq 1 + N(h-1) + N(h-2)$

$\rightarrow N(h) \geq 1 + 2N(h-2)$
 $\geq 1 + 2 + 4N(h-4)$
 \dots
 $\geq 1 + 2 + \dots + 2^{i-1} + 2^i N(h-2i)$

$\rightarrow N(h) \geq 2^i - 1 + 2^i N(h-2i) (*)$


When $h-2i \leq 0 \rightarrow i = \frac{h}{2}$, $(*)$ becomes:

$\rightarrow N(h) \geq 2^{h/2} - 1 + 2^{h/2} N(0)$
 $\rightarrow N(h) \geq 2^{h/2+1} - 1$

- Where $N(h) = n \rightarrow h \leq 2\log(n+1) - 1$
- Therefore. $h = O(\log n)$

www.ctu.edu.vn
12

12




Declaration

- Similar to BST.
- But each node has some more attributes to support the balancing of the tree.
 - Height
- Declaration

```
typedef struct NodeTag{
    KeyType key;
    struct NodeTag* left;
    struct NodeTag* right;
    int height;
}Node;
typedef Node *Tree;
```

www.ctu.edu.vn 13

13



Agenda

- Introduction
- Rotations
- Operators
- Summary

www.ctu.edu.vn 14

14



Rotations

- Insertion or deletion could cause the tree imbalanced.
- Rotation is a local operator to change the structure of the tree but keep node orders such that the tree becomes more balanced.
- 4 rotations
 - Left rotation
 - Right rotation
 - Left Right rotation
 - Right Left rotation

www.ctu.edu.vn

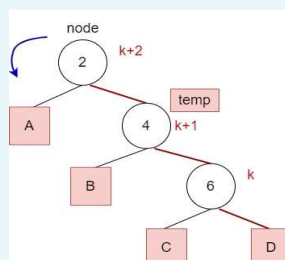
15

15

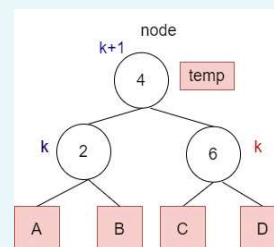


Left rotation

- Right - Right case: violation in Right sub tree, Right sub tree
- Left rotate the node around temp



Left rotate



www.ctu.edu.vn

16

16

Steps of left rotation

```

leftRotate(node):
    temp = node->right
    node->right = temp->left
    temp->left = node
    calculateHeight(node)
    calculateHeight(temp)
    return temp

```

$T(n) = O(1)$

www.ctu.edu.vn

17

Right rotation

- Left - left case: violation in Left sub tree, Left sub sub tree
- Right rotate the node around temp

Right rotate

www.ctu.edu.vn

18

Steps for Right rotation

```

rightRotate(node):
    temp = node->left
    node->left = temp->right
    temp->right = node
    calculateHeight(node)
    calculateHeight(temp)
    return temp

```

$T(n) = O(1)$

www.ctu.edu.vn

19

Left Right rotation


- Left – Right case: violation in Left sub tree, Right sub tree
- 2 rotations
 - Left rotate temp around z
 - Right rotate node around temp

Left Rotate

Right Rotate

www.ctu.edu.vn

20



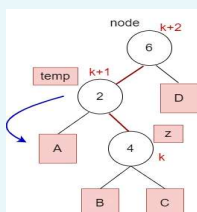
Steps for Left Right rotation

```

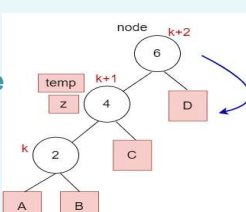
leftrightRotate(node):
    node->Left = leftRotate(node->left)
    return rightRotate(node)

```

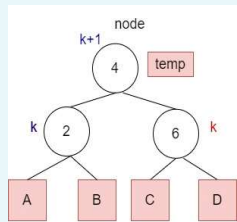
$T(n) = O(1)$



Left Rotate




Right Rotate



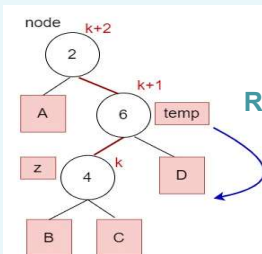
www.ctu.edu.vn

21

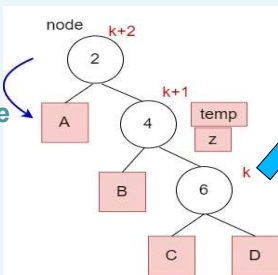


Right Left rotation

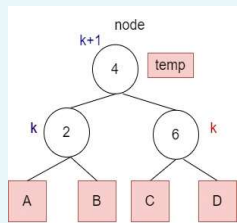
- Right - Left case: violation in Right sub tree, Left sub tree
- 2 rotations
 - Right rotate temp around z
 - Left node around temp



Right Rotate




Left Rotate



www.ctu.edu.vn

22

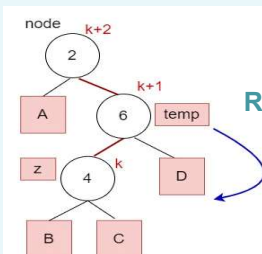


Steps for Right Left rotation

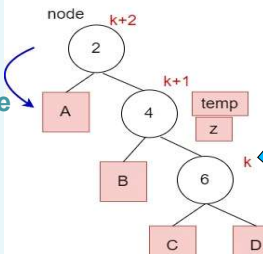
```

rightleftRotate(node):
    node->right = rightRotate(node->right)
    return leftRotate(node)
        
```

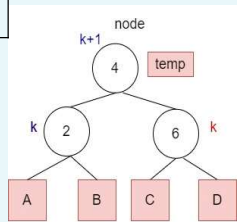
$T(n) = O(1)$



Right Rotate



Left Rotate



www.ctu.edu.vn 23

23



Agenda

- Introduction
- Rotations
- Operations
- Summary

www.ctu.edu.vn 24

24



Insert to an AVL tree

- Insert a node as **BST insertion**. Let the inserted node be w
- From w , move **backward** to the root to find the **first imbalanced node** ($|BF| > 1$)
 - Let z be the **first imbalanced node**
 - Let y be z 's **child** on the path from w to z
 - Let x be y 's **child** on the path from w to z
- Choose rotation for balancing the node z
 - y is **left child** of z , x là **left child** of y → **Right rotate** z
 - y is **left child** of z , x is **right child** of y → **Right Left rotate** z
 - y is **right child** of z , x is **right child** of y → **Left rotate** z
 - y is **right child** of z , x is **left child** of y → **Right rotate** z



Insert to an AVL tree

- In all cases, after **rotation at z** , the whole tree becomes **balanced**.

```

insertNode(node, k):
    BSTInsert(node, k)
    calculateHeight(node)
    balance = getBalance(node)
    if (balance > 1):
        if (k < node->left->key): return rightRotate(node)
        else: return leftRightRotate(node)
    if (balance < -1):
        if (k > node->right->key): return leftRotate(node)
        else: return rightLeftRotate(node)
    return node
  
```

$T(n) = O(\log n)$

Example

■ Insert 10

The diagram illustrates the insertion of 10 into a B+ tree. The initial tree has root 50 with children 30 and 80. Node 30 has child 20. Inserting 10 as a new leaf under 20 results in a Left Left case where the balance factor (BF) of node 30 is 2. A right rotation is performed around node 30, making 20 the new root of this subtree, with 10 as its left child and 30 as its right child. The final balanced tree has root 50 with children 20 and 80. Node 20 has children 10 and 30. All nodes now have a balance factor of 1 or 0.

Left Left case
30: BF = 2

Right rotate

27

www.ctu.edu.vn

27

Example

■ Insert 120

The diagram illustrates the insertion of 120 into a B+ tree. The initial tree has root 50 with children 20 and 80. Node 20 has children 10 and 30. Node 80 has child 100. Inserting 120 as a new leaf under 100 results in a Right Right case where the balance factor (BF) of node 80 is -2. A left rotation is performed around node 80, making 100 the new root of this subtree, with 80 as its left child and 120 as its right child. The final balanced tree has root 50 with children 20 and 100. Node 20 has children 10 and 30. Node 100 has children 80 and 120. All nodes now have a balance factor of 1 or 0.

Right Right case
80: BF = -2

Left rotate

28

www.ctu.edu.vn

28

Example
■ Insert 40

Left Right case
50: BF = 2

Left rotate 20

Right rotate 50

www.ctu.edu.vn

29

Example
■ Insert 45

Right Left case
30: BF = -2

Right rotate 50

Left rotate 30

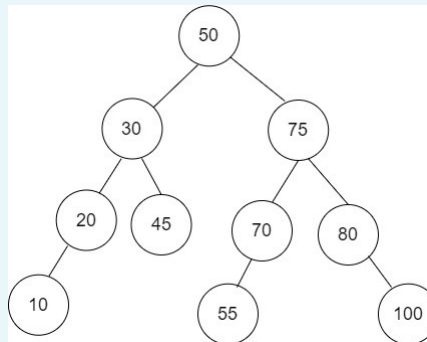
www.ctu.edu.vn

30



Example

- Draw an AVL from the list
50, 70, 30, 10, 20, 45, 80, 75, 100, 55



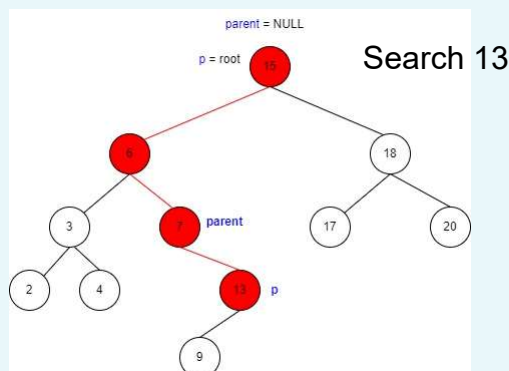
www.ctu.edu.vn 31

31



Search a node in an AVL tree

- Similar to searching on BST
- $T(n) = O(\log n)$



www.ctu.edu.vn 32

32



Delete a node in an AVL tree

- Let the node to be deleted be w
- Process **BST Deletion**.
- From w , move backward to the root to find the first **imbalanced node**. Let z be that node, y be a **child** of z which has greater height, x be a **child** of y which has greater height.
- 4 cases
 - y is left child of z , x is left child of y → Right rotate z
 - y is left child of z , x is right child of y → Left Right rotate z
 - y is right child of z , x is right child of y → Left rotate z
 - y is right child of z , x is left child of y → Right Left rotate z
- After balancing at z , it's possible to do balancing **some ancestors** of z
- $T(n) = O(\log n)$

www.ctu.edu.vn

33

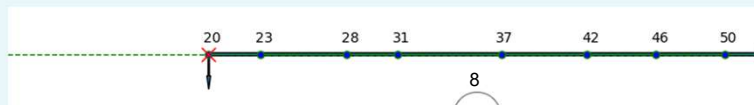
33



Recall Runway reservation system

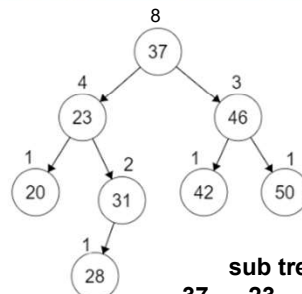
- Pending events

$R = \{37, 23, 20, 46, 31, 42, 28, 50\}$



- Operators

- $\text{check}(t, k): O(\log n)$
- $\text{register}(t, k): O(\log n)$
- $\text{remove}(t): O(\log n)$
- $\text{rank}(t): O(\log n)$




Keep track the size of sub trees during insertion and deletion

sub tree
 $37 \quad 23 \quad 46 \quad 42$
 $1 + 4 + 1 + 1 = 7$
 sub tree

www.ctu.edu.vn

34

34




Agenda

- Introduction
- Rotations
- Operations
- Summary

www.ctu.edu.vn 35

35



Summary

- A BST where the difference between the height of left child & right child of any node does not exceed 1.
- Complexity

	Search	Insert	Delete
BST	$O(h)$	$O(h)$	$O(h)$
AVL	$O(\log n)$	$O(\log n)$	$O(\log n)$

www.ctu.edu.vn 36

36

