CANTHO UNIVERSITY

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

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

1

CANTHO UNIVERSITY

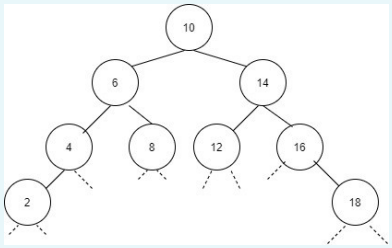
Agenda

- Introduction
- Rotations
- Operators
- Summary

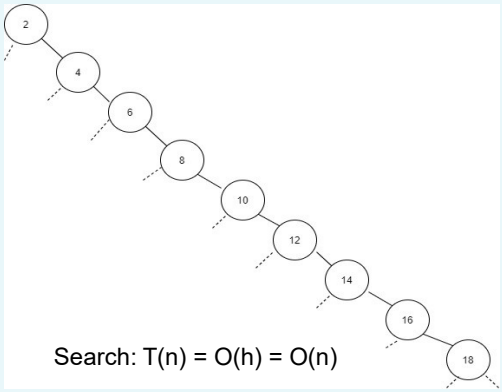
2

Balanced problem

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



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



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

Balanced binary search tree

- A BST is balanced if h = O(logn) (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
  - …

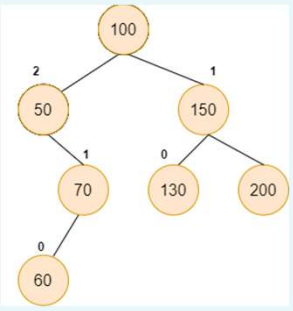
CANTHO UNIVERSITY

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) = height(left(p)) - height(right(p))
- Recall: The height of a node is the length of the path from it to the farthest descendant leave node.
  - A NULL node has the length -1
  - A laeave node has the length 0
  - height(p) = 1 + **MAX**(height(**left(p)**), height(**right(p)**))
- A node is called height-balanced if its balanced factor is either -1, 0, or 1

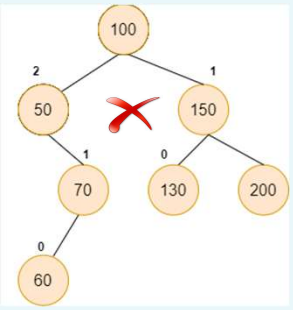
CANTHO UNIVERSITY

Examples



CANTHO UNIVERSITY

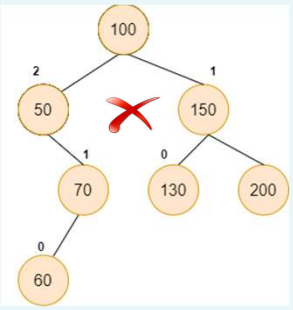
Examples

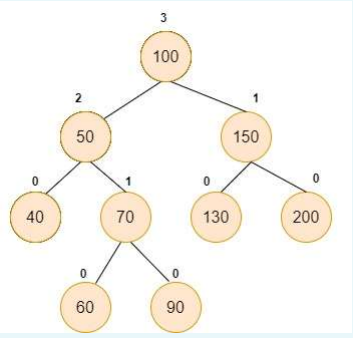


7

CANTHO UNIVERSITY

Examples

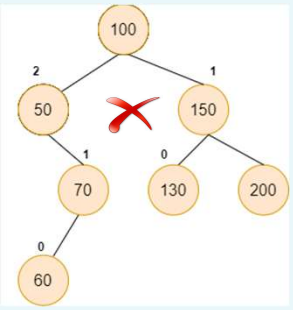


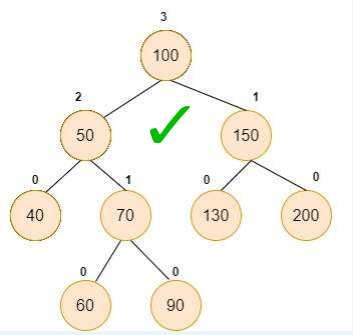


8

CANTHO UNIVERSITY

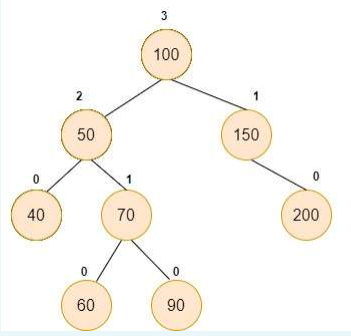
Examples





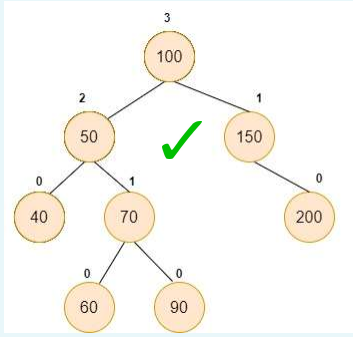
CANTHO UNIVERSITY

Examples



CANTHO UNIVERSITY

Examples



CANTHO UNIVERSITY

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) >= 1 + N(h-1) + N(h-2)

→ N(h) >= 1 + 2N(h-2)

>= 1 + 2 + 4N(h-4)

…

>= 1 + 2 + … + 2^{i-1} + 2^iN(h-2i)

→ N(h) >= 2^i - 1 + 2^iN(h-2i) (*)

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

→ N(h) >= $2^{h/2}$ - 1 + $2^{h/2}$N(0)

→ N(h) >= $2^{h/2+1}$ - 1

- Where N(h) = n → h <= 2log(n+1) – 1
- Therefore, h = O(logn)

CANTHO UNIVERSITY

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;
```


CANTHO UNIVERSITY

Agenda

- Introduction
- Rotations
- Operators
- Summary

CANTHO UNIVERSITY

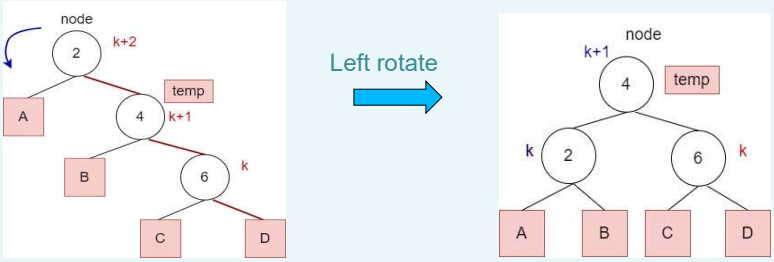
Rotations

- 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

CANTHO UNIVERSITY

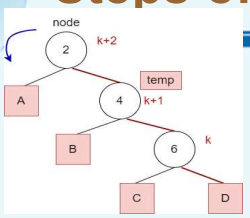
Left rotation

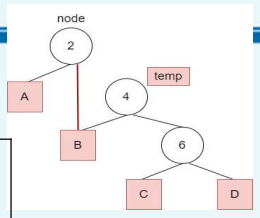
- Right - Right case: violation in Right sub tree, Right sub sub tree
- Left rotate the node around temp



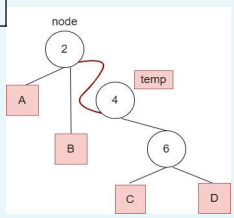
CANTHO UNIVERSITY

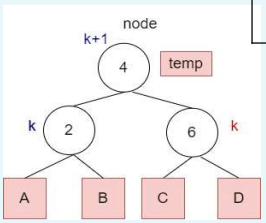
Steps of left rotation






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



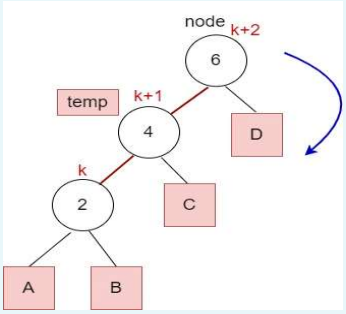


T(n) = O(1)

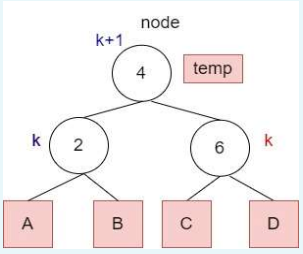
CANTHO UNIVERSITY

Right rotation

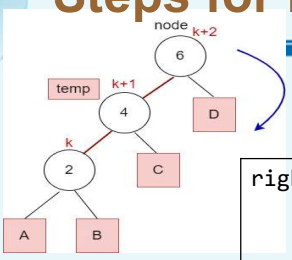
- Left - left case: violation in Left sub tree, Left sub sub tree
- Right rotate the node around temp



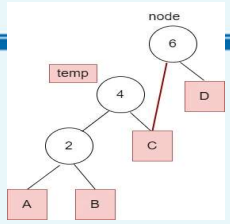
Right rotate

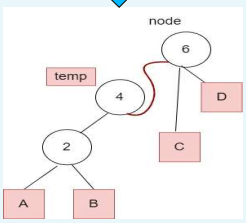


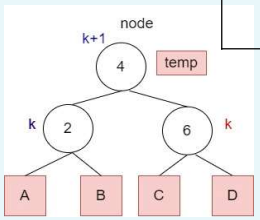
Steps for Right rotation




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



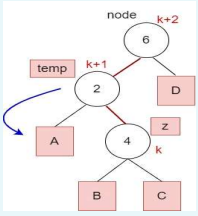




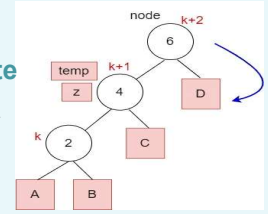
T(n) = O(1)

Left Right rotation

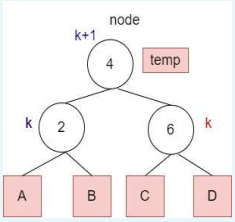
- Left – Right case: violation in Left sub tree, Right sub sub tree
- 2 rotations
  - Left rotate temp around z
  - Right rotate node around temp



Left Rotate



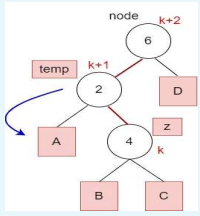
Right Rotate



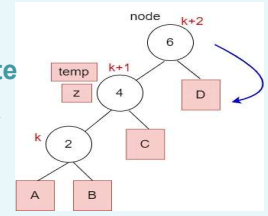
Steps for Left Right rotation

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

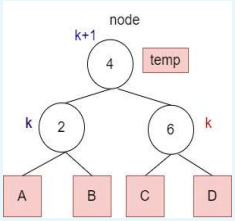
T(n) = O(1)



Left Rotate

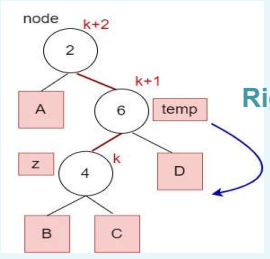


Right Rotate

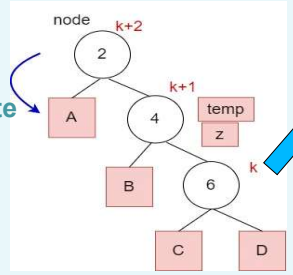


Right Left rotation

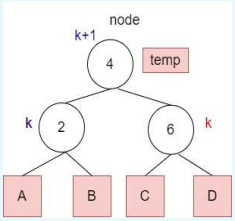
- Right - Left case: violation in Right sub tree, Left sub sub tree
- 2 rotations
  - Right rotate temp around z
  - Left node around temp



Right Rotate



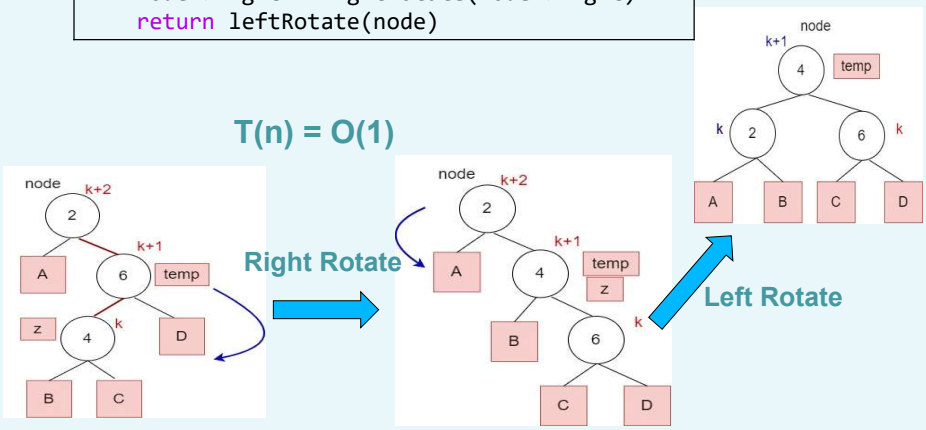
Left Rotate



Steps for Right Left rotation

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

T(n) = O(1)



Agenda

- Introduction
- Rotations
- Operations
- Summary

CANTHO UNIVERSITY

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

CANTHO UNIVERSITY

Insert to an AVL tree

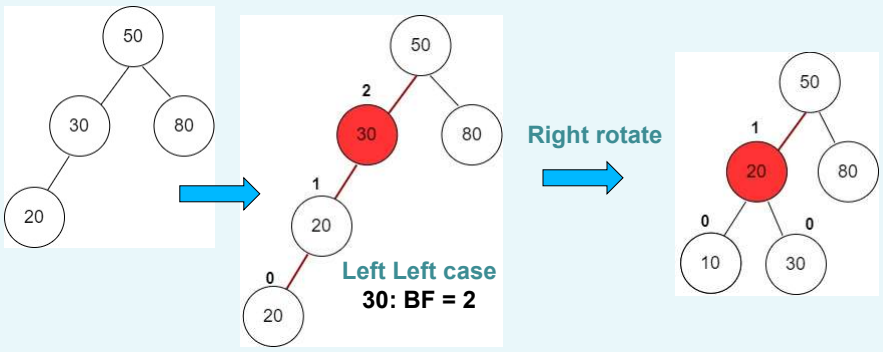
- In all cases, after rotation at z, the whole tree becomes balanced.

```
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(logn)

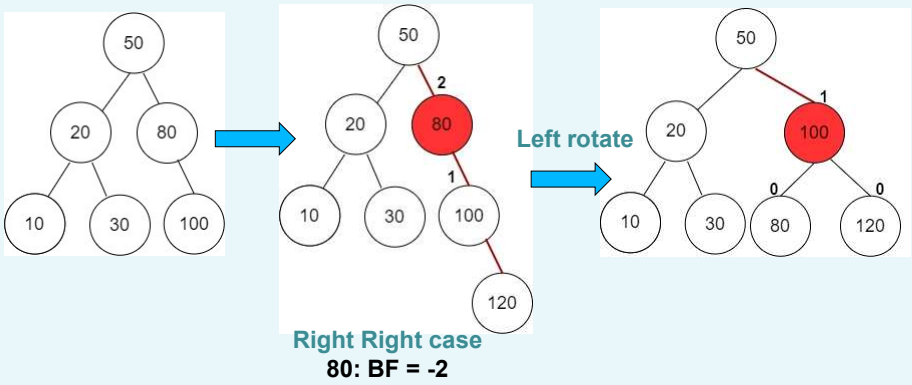
Example

- Insert 10



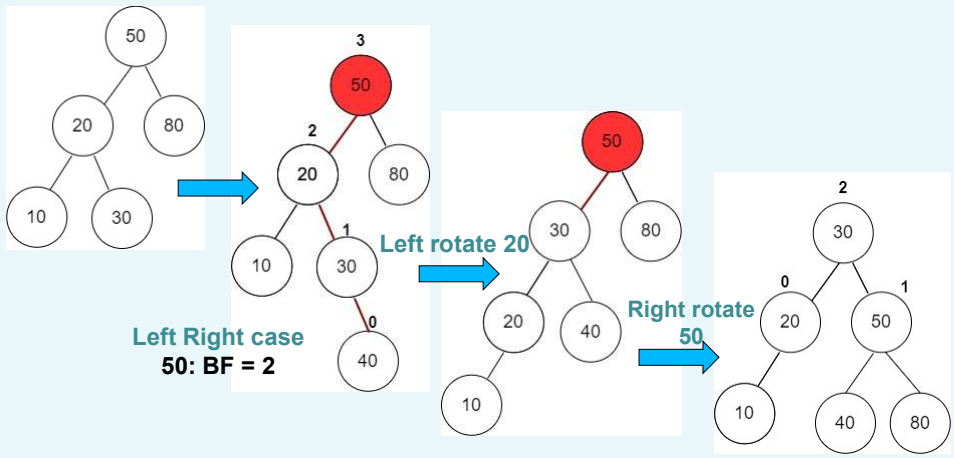
Example

- Insert 120



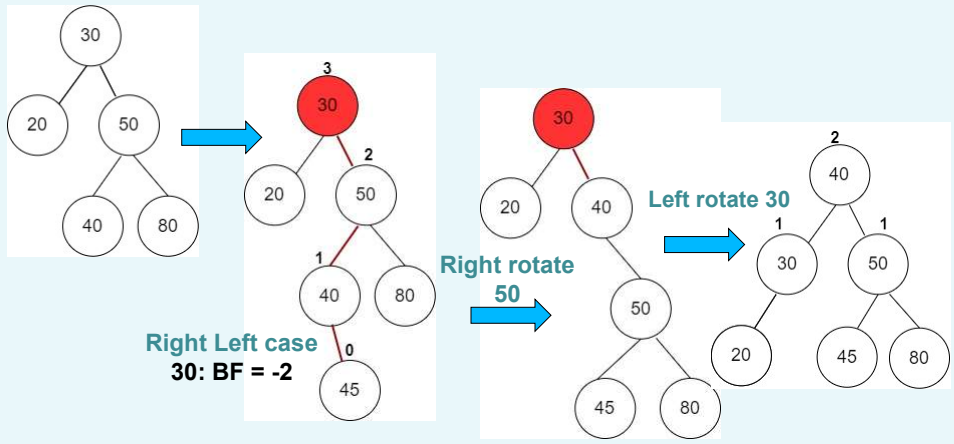
Example

- Insert 40



Example

- Insert 45

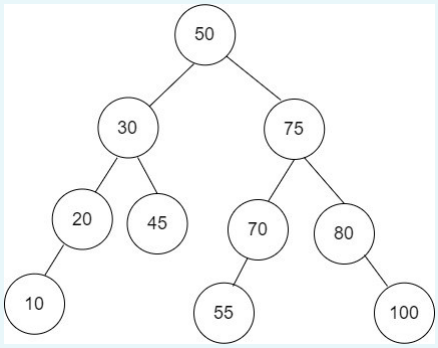


CANTHO UNIVERSITY

Example

- Draw an AVL from the list

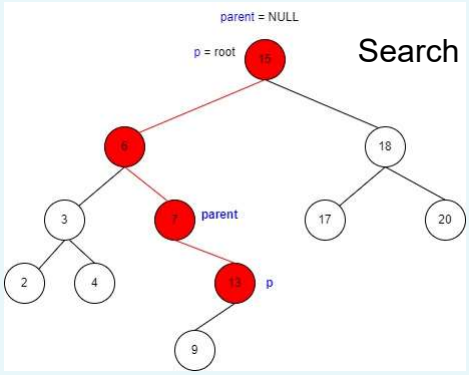
  50, 70, 30, 10, 20, 45, 80, 75, 100, 55



CANTHO UNIVERSITY

Search a node in an AVL tree

- Similar to searching on BST
- T(n) = O(logn)



Search 13

Delete a node in an AVL tree

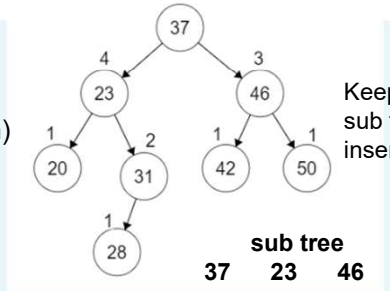
- Let the node to be deleted be w
- Process BST Deletion.
- From w, move backward to the root to find the first imbalanced node. Let z be that node, y be a child of z which has greater height, x be a child of y which has greater height.
- 4 cases
  - y is left child of z, x is left child of y → Right 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(logn)

Recall Runway reservation system

- Pending events

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

- Operators
  - check(t, k): O(logn)
  - register(t, k): O(logn)
  - remove(t): O(logn)
  - rank(t): O(logn)



Keep track the size of sub trees during insertion and deletion

sub tree

37 23 46 42

1 + 4 + 1 + 1 = 7

sub tree

CANTHO UNIVERSITY

Agenda

- Introduction
- Rotations
- Operations
- Summary

CANTHO UNIVERSITY

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(logn)	O(logn)	O(logn)

CANTHO UNIVERSITY

Q&A

www.ctu.edu.vn

37