

EECS 204002
Data Structures 資料結構
Prof. REN-SONG TSAY 蔡仁松 教授
NTHU

CH. 10 EFFICIENT BINARY SEARCH TREES

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 1

Binary Search Trees

- All BST operations complexity = $O(h)$
 - h = height of the BST
- Worst case: $h = n$
 - Ex: insert keys 1, 2, ..., n
- Best case: $h = \log n$
 - Ex: insert keys 4, 2, 6, 1, 3, 5, 7

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 2

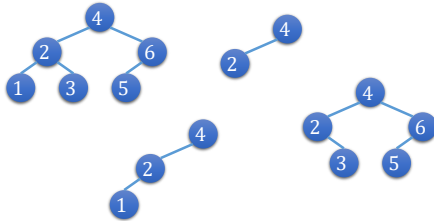
What is the Best Case?

- If BST retains a complete tree
- But expensive to retain a complete tree
 - Ex: insert 3 into the tree on the left

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 3

A Compromise

- Fairly, but not perfectly, balanced tree
 - Depths of the left and right subtrees $\Rightarrow \pm 1$
- Which one is “balanced”?



2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 4

How to Keep a Balanced BST ?

- AVL Trees
- Red-black Trees (self-study)
- Splay trees (self-study)
 - Self adjusting trees
- B-trees
 - Multiway search trees

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 5

10.2

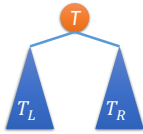
AVL Trees

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 6

10.2 **Height Balanced Trees**

- An empty tree is height balanced.
- If T is a non-empty binary tree with T_L and T_R
 - As its left and right subtrees respectively
- **Balance factor**

$$bf(T) = height(T_L) - height(T_R)$$
- T is height balanced iff
 - 1) T_L and T_R are height balanced.
 - 2) $|bf(T)| \leq 1$




2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 7

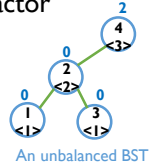
10.2 **Definition of AVL Trees**

- AVL tree is a *height-balanced* binary search tree. (**A**delson, **V**elskii, **L**andis)
- Each node in an AVL tree stores the current node height for calculating the balance factor


Balance factor



Representation



An unbalanced BST



An AVL Tree

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 8

10.2 **Prove $N \geq 2^{h/2}$**

for an N -node AVL tree of height h

- Idea: to prove by showing $N \geq N(h) \geq 2^{h/2}$, where $N(h)$ = minimum # of nodes of an h -height AVL tree.
- Induction
 - $N(1) = 1, N(2) = 2$
 - $N(h) = N(h-1) + N(h-2) + 1$
 - $N(h) \geq 2 \cdot N(h-2)$
- Solution
 - $N(h) \geq 2N(h-2) \geq 2(2N(h-4)) \geq 2^i N(h-2i) \approx 2^{h/2}$
 - Or $h = O(\log_2 N)$

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 9

10.2 **Rebalancing Process**

- BST insertion/deletion operation may cause nodes with balance factor > 1 or < -1 .
- Rebalancing process
 - Update the heights (balance factors) from the inserted/deleted node up to the root.
 - Fix unbalanced situations by rotations.

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 10

Rebalancing Operations

Need two rotations
Right rotation on selected area
Then, left rotation

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 11

Rebalancing Operations

Need two rotations
Left rotation on selected area
Then, right rotation

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 12

10.2 **4 Unbalanced Situations**

- 2 outside cases: require single rotation (LL, RR)
- 2 inside cases: require double rotation (LR, RL)

2 outside cases 2 inside cases

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 14

Outside RR Case - Left Rotation

- The new node is inserted in the **right** subtree of the **right** subtree of A

Left Rotation

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 15

Outside LL Case - Right Rotation

- The new node is inserted in the **left** subtree of the **left** subtree of A

Right Rotation

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 16

Inside RL Case - LR Rotation

- The new node is inserted in the right subtree of the left subtree of A
- Left rotation + Right rotation

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 17

Inside LR Case - RL Rotation

- The new node is inserted in the left subtree of the right subtree of A
- Right rotation + left rotation

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 18

10.2 ADT: AVL Tree

```

template < class T > class AVLTree;

template < class T >
class TreeNode {
friend class AVLTree <T>;
private:
    T data;
    int height;
    void updateHeight();
    int bf();
    TreeNode<T>* left, right;
};

template <class T>
class AVLTree{
public:
    // Constructor
    AVLTree(void) {root=NULL;}
    // Tree operations here...
private:
    TreeNode<T> *root;
};
    
```

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 19

10.2 **AVL Tree: Example: Insert 17**

LL Case Rebalancing
Right rotation!

Update height and bf

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 23

10.2 **AVL Tree: Example: Insert 17**

Finish checking

Update height and bf

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 24

10.2 **AVL Tree: Example: Insert 5**

RL Case Rebalancing
LR rotations!

Update height and bf

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 25

10.2 **AVL Tree: Example: Insert 5**

After left rotation
continue rebalancing, right rotation!

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 26

10.2 **AVL Tree: Example: Insert 5**

Finish checking

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 27

10.2 **AVL Tree: Example: Delete 16**

14 replaces 16
Delete 16!

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 28

10.2 **AVL Tree: Example: Delete 16**

RR Case Rebalancing
Left rotation!

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 29

10.2 **AVL Tree: Example: Delete 16**

Finish checking

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 30

EECS 204002
Data Structures 資料結構
Prof. REN-SONG TSAY 蔡仁松 教授
NTHU

CH. 11 MULTIWAY SEARCH TREES

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 53

11.2

B Trees

2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 54

B-tree: Definition

A B-tree of order m is a height-balanced m -way search tree, where each node may have up to m children, and in which:

1. Each internal node contains no more than $m-1$ keys
2. All leaves are on the same level
3. All nodes except the root have $\lceil m/2 \rceil$ to m children
4. The root is either a leaf node, or it has 2 to m children
5. m usually is odd.

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 55

2-3 Trees

- A B-Tree of order 3 is called a 2-3 Tree.
 - 2 to 3 pointers
- In a 2-3 tree, each internal node has either 2 or 3 children.
- Most practical applications adopt larger order (e.g., $m = 128$) B-Trees.

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 56

Example of a 2-3 Tree

$m = 3$
of Children: 2-3

Insert 70?

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 57

B-tree: Insert

- Search
- Insert the new key into a leaf, which is the node under work
- If the node overflows
 - If it is root, create a new root as its parent
 - Split the node into two and push up the middle key to the node's parent
 - Let the parent node be the node under work and repeat the overflow checking and split process.
- Done, if no overflow.

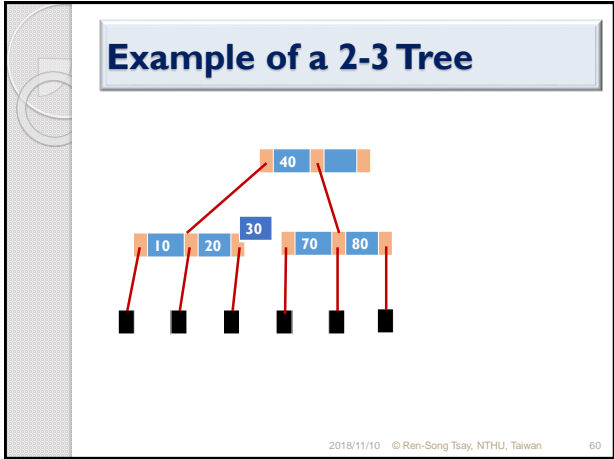
2018/11/9 © Ren-Song Tsay, NTHU, Taiwan 58

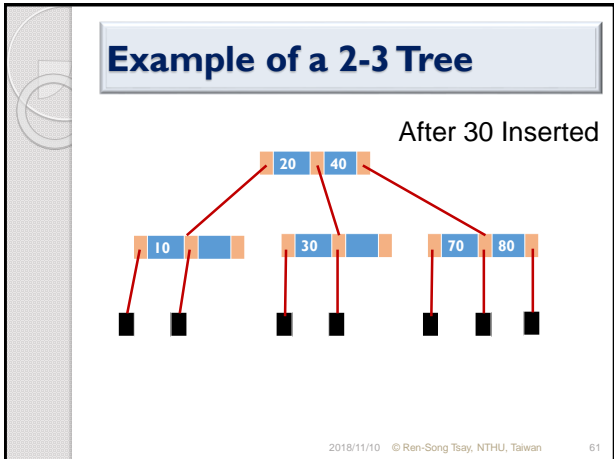
Example of a 2-3 Tree

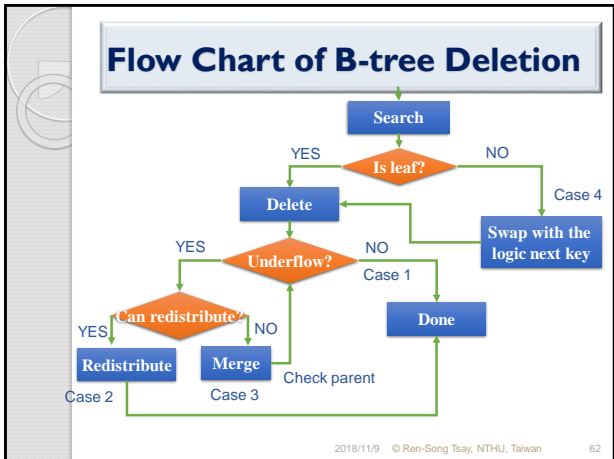
After 70 Inserted

Insert 30?

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 59







Deletion from a 2-3 Tree

Delete 70?

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 63

Deletion from a 2-3 Tree

After 70 deleted
Case 1

Delete 10?

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 64

Deletion from a 2-3 Tree

After 10 deleted
Case 1

Delete 60?

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 65

Deletion from a 2-3 Tree

After 60 deleted
Case 2

Delete 95?

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 66

Deletion from a 2-3 Tree

After 95 deleted
Case 3

Delete 50?

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 67

Deletion from a 2-3 Tree

After 50 deleted
Case 1

Delete 20?

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 68

Deletion from a 2-3 Tree

After 20 deleted

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 69

B-Tree Exercise

Insert the following keys to a 5-way B-tree:
 3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4,
 31, 35, 56

Then delete all nodes subsequently in the reverse order of the insertion.

2018/11/10 © Ren-Song Tsay, NTHU, Taiwan 72
