CSCI 335
Software Design and Analysis III
Lecture 9: AVL Trees

Professor Anita Raja

09-29-22

1

1

Agenda

- IPL and BST Average case analysis
- AVL Trees
  - Insertion
  - Deletion
  - Amortized Cost
- HW2 Discussion

2

2

Balanced Trees

- Try to balance after tree operations and make operations logarithmic
  - AVL tree (Balance is always preserved)
  - Splay tree (Self-adjusts towards balance)

3

3

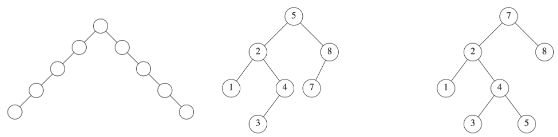
AVL trees

- Adelson, Velskii and Landis
- Binary Search Trees with a balance condition
  - Must be easy to maintain
  - Balance ensures depth is O(logN)
- For all the nodes in the tree the height of the left subtree minus the height of the right subtree must be 0, 1 or -1.
- Assume that the height of empty subtree is -1.
  - Height is at most 1.44log(N+2)-1.328 but in practice it is only slightly more than logN.

4

4

Which one is AVL?



5

AVL trees

- Minimum number of nodes of AVL tree of height h:
  S(h) = S(h – 1) + S(h - 2) + 1
  S(0)=1, S(1) = 2
- S(h) closely related to Fibonacci numbers!

6

AVL trees

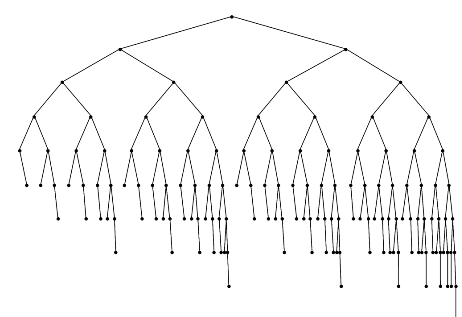
- Minimum number of nodes of AVL tree of height h:

$$S(h) = S(h-1) + S(h-2) + 1$$
$$S(0) = 1$$
$$S(1) = 2$$

- Why?
- Can prove height properties since this is similar to Fibonacci numbers. Number of nodes grows exponentially as height increases.
- Roughly $1.44 \log(N+2) - 1.328$
- Generally only a little more than $\log N$

7

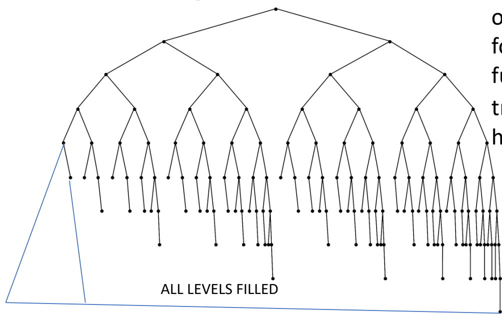
AVL trees (smallest tree of height 9)



8

AVL trees: larger number of nodes for tree of height 9

$N = 2^{h+1} - 1$: number of nodes for full tree of height h



9

9

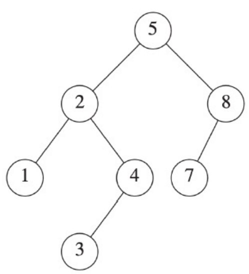
Insertion cases

- After insertion only nodes on the path between insertion point and the root may have their balance altered.
- Suppose that α is the first node on the path that needs to be rebalanced.
- Four cases of violation. Insertion into:
  1. Left subtree of left child of α
  2. Right subtree of left child of α
  3. Left subtree of right child of α
  4. Right subtree of right child of α
- Rotation operation at α will balance the tree.

10

10

Exercise: Insert 4.5 in the AVL tree



11

11

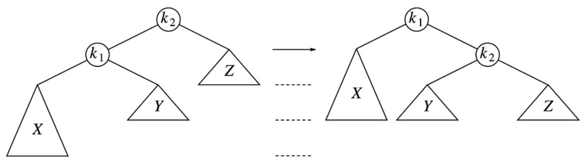
Insertion cases

- 1,4 (left-left or right-right) : Single rotation
- 2,3 (right-left or left-right) : Double rotation
- Implementation of double rotation simply involves two calls to the single rotation function.
- Conceptually it may be easier to think about it as a different operation.

12

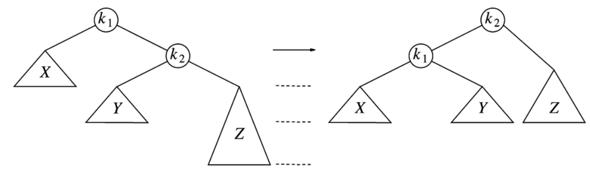
12

Single rotation (case 1)



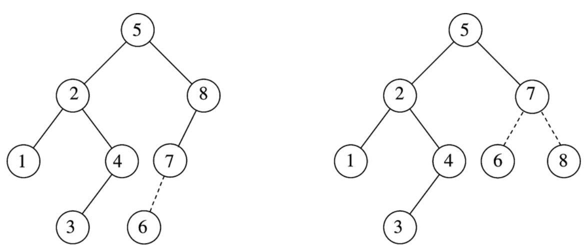
13

Single rotation (case 4)



14

Example, case 1



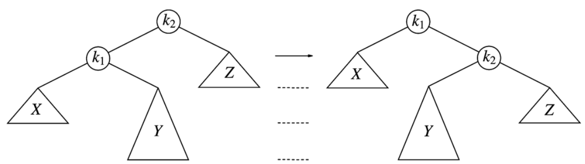
15

Single rotation example

- Insert: 3,2,1
- Insert: 4,5,6,7
- AVL animation: https://www.cs.usfca.edu/%7Egalles/visualization/AVLtree.html

16

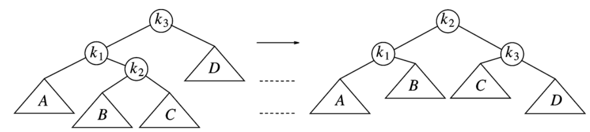
Single rotation fails in this case



17

17

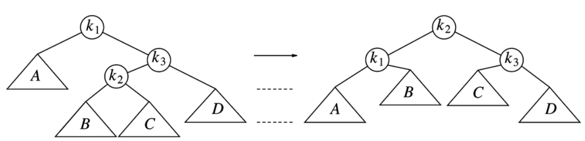
Double rotation (left-right) (case 2)



18

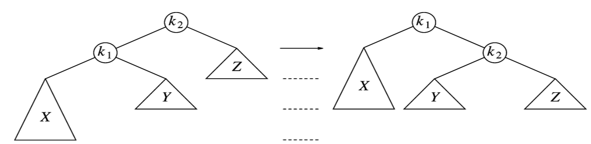
18

Double rotation (right-left) (case 3)



19

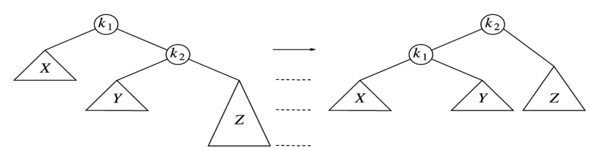
19



Case 1 (single)

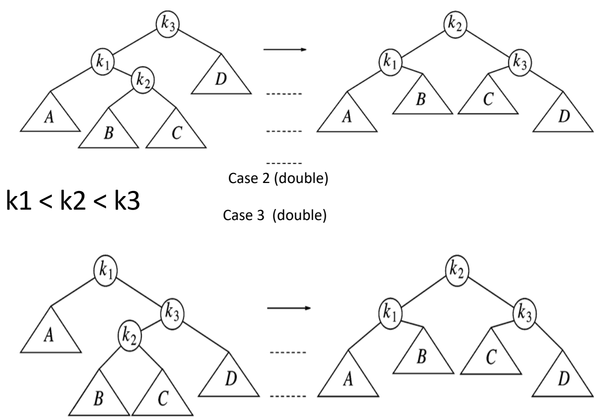
k1 < k2

Case 4 (single)

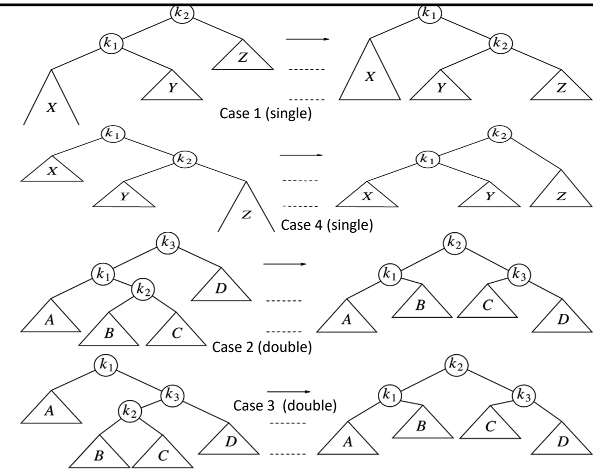


20

20



21



22

Example

- Continue by inserting: 16, 15, 14, 13, 12, 11, 10
- Insert 8,9.

23

23

AVL Implementation : AVL node

```
// AVL Tree implementation
// Usage: AvlTree<int> a_tree;
// a_tree.Insert(10);
template <typename Comparable>
Class AvlTree {
public: // … Big five.
private:
struct AvlNode {
Comparable element_;
AvlNode *left_;
AvlNode *right_;
int height_;
AvlNode(const Comparable &the_element, AvlNode *lt, AvlNode *rt, int h = 0):
element_{the_element}, left_{lt}, right_{rt}, height_{h} { }
AvlNode(Comparable &&the_element, AvlNode *lt, AvlNode *rt, int h = 0):
element_{std::move(the_element)}, left_{lt}, right_{rt}, height_{h} { }
};
AvlNode *root_;
// Returns height of subtree with root t.
int height(const AvlNode *t) const {
return t == nullptr ? -1: t->height_;
}
};
```

24

24

Implementation: Insert

```
// Internal method to insert into an AVL subtree.// x is the item to insert. // t is the node that roots the subtree.
// t's height maybe updated. If there is a violation at t, a rotation (single or double)
// will be performed.
void Insert(const Comparable &x, AvlNode * &t) {
if (t == nullptr)
t = new AvlNode{x, nullptr, nullptr, 0};
else if (x < t->element) {
Insert(x, t->left);
if (height(t->left_) – height(t->right_) == 2) {
if (x < t->left_->element_)
RotateWithLeftChild(t);
else
DoubleWithLeftChild(t);
}} else if (t->element_ < x) {
insert(x, t->right_);
if (height(t->right_) – height(t->left_) == 2) {
if (t->right->element_ < x)
RotateWithRightChild(t);
else
DoubleWithRightChild(t);
} // else, duplicate; do nothing
t->height_ = Max(height(t->left_), height(t->right_)) + 1;
}
```

25

25

Implementation: Alternative Insert

```
// ALTERNATIVE insert.
Void Insert(const Comparable &x, AvlNode * & t)
{
if (t == nullptr)
t = new AvlNode{x, nullptr, nullptr, 0};
else if (x < t->element_)
Insert(x, t->left_);
else if (t->element_ < x)
Insert(x, t->right_);
balance(t);
}
```

26

26

Implementation: Balance

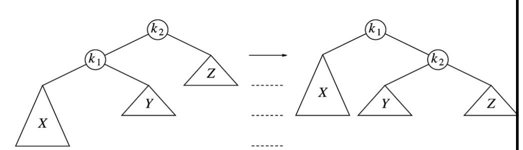
```
// Assume t is balanced or within one of being balanced.
// If node t is not balanced, AVL
// rotations will restore balance.
void balance(AvlNode *&t) {
if (t == nullptr)
return;
if (height(t->left_) – height(t->right_) > 1) {
if (height(t->left_->left_) >= height(t->left_->right_))
RotateWithLeftChild(t);
else
DoubleWithLeftChild(t);
} else if (height(t->right_) – height(t->left_) > 1) {
if (height(t->right_->right_) >= height(t->right_->left_))
rotateWithRightChild(t);
else
doubleWithRightChild(t);
}
t->height_ = Max(height(t->left_), height(t->right_)) + 1;
}
```

27

27

Implementation: Rotate with left child

```
// Rotate binary tree node with left child.
// For AVL trees, this is a single rotation for case 1.
// Update heights, then set new root.
Void RotateWithLeftChild(AvlNode * &k2) {
AvlNode *k1 = k2->left_;
k2->left_ = k1->right_;
k1->right_ = k2;
k2->height_ = Max(height(k2->left_),
height(k2->right_)) + 1;
k1->height_ = Max(height(k1->left_),
k2->height_) + 1;
k2 = k1;
}
```

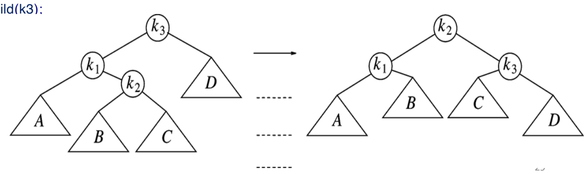


28

28

Implementation: Double rotate with left child

```
// Double rotate binary tree node: first left child.
// with its right child; then node k3 with new left
child.
// For AVL trees, this is a double rotation for case 2.
// Update heights, then set new root.
void DoubleWithLeftChild(AvlNode * &k3) {
        RotateWithRightChild(k3->left_);
        RotateWithLeftChild(k3);
}
```



29

AVL Tree Deletion

- What could happen to the tree balance when we perform a remove?
- Can we simply add balance(t)to the BST implementation of remove?

30

Amortized cost

- Consider a sequence of M operations (insert/delete/find):
- Suppose that total cost is **O(M * f(N))** irrespective of the actual sequence of operations.
- The average cost is **O(f(N))** for each operation.
- This is called **amortized running time**.
- Caveat:

Individual operations in the sequence can be expensive though !

31

Amortized cost

- Worst-case bound on sequence of operations
- Consider a sequence of M operations(insert/delete/find)
  - What is the total cost?
  - What is the average cost per operation?

32

Amortized cost

- Consider a sequence of M operations (insert/delete/find):
- Example: binary search tree (regular - unbalanced)
  - M operations can cost in the worst case **O(M * N)**
  - Amount of work proportional to height of tree – maintain ordering property of BST for all subtrees.
  - Each operation will **not** cost more than **O(N)**
  - On average each operation costs **O(N)**

33

33

Amortized cost

- Consider a sequence of M operations (insert/delete/find):
- AVL tree:
  - A sequence of M operations will cost **O(M * logN)**
  - Each operation will **not** cost more than **O (logN)**
  - On average each operation costs **O (logN)**

34

34

AVL summary

- What is the cost of
  - Search
  - Insertion
  - Deletion

  in an AVL tree of N nodes?
- Exercise: Work on the 2 AVL sequences, understand the details and make the topic your own.

35

35

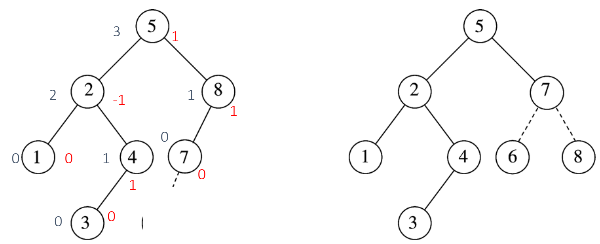
Next week

- Splay Trees, Btrees, Sets, Maps
- Work on HW2
- HW1 grades/feedback released this weekend
- Keep up with lecture notes/readings

36

36

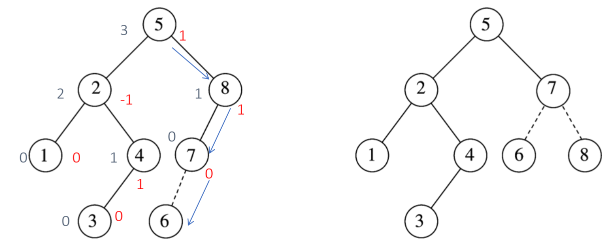
Example, case 1 (insert 6)



37

Example, case 1 (insert 6)

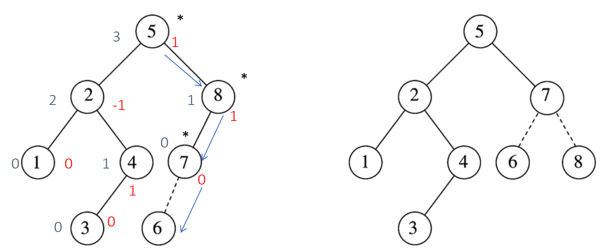
VIOLATIONS OF BALANCE POSSIBLE ONLY AT PATH OF INSERTION



38

Example, case 1 (insert 6)

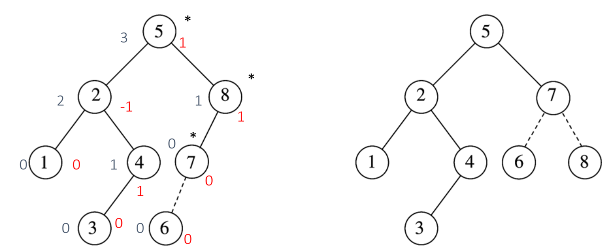
VIOLATIONS OF BALANCE POSSIBLE ONLY AT PATH OF INSERTION



39

Example, case 1 (insert 6)

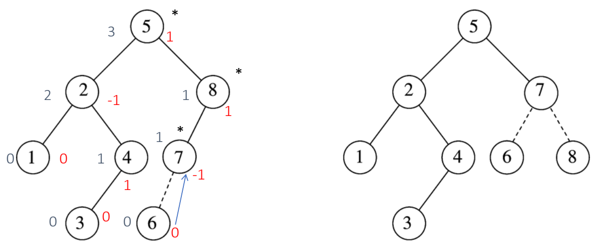
WALK IN OPPOSITE DIRECTION – UPDATE HEIGHTS AND CHECK FOR VIOLATIONS



40

Example, case 1 (insert 6)

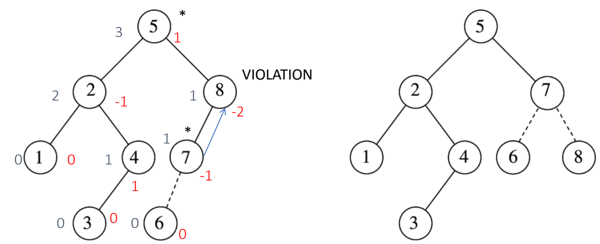
WALK IN OPPOSITE DIRECTION – UPDATE HEIGHTS AND CHECK FOR VIOLATIONS



41

Example, case 1 (insert 6)

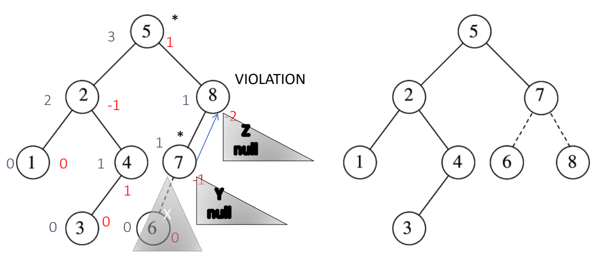
WALK IN OPPOSITE DIRECTION – UPDATE HEIGHTS AND CHECK FOR VIOLATIONS



42

Example, case 1 (insert 6)

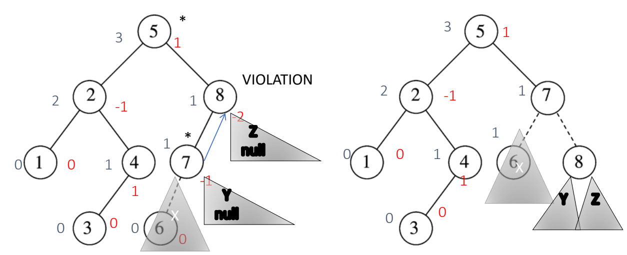
WALK IN OPPOSITE DIRECTION – UPDATE HEIGHTS AND CHECK FOR VIOLATIONS



43

Example, case 1 -> single rotation

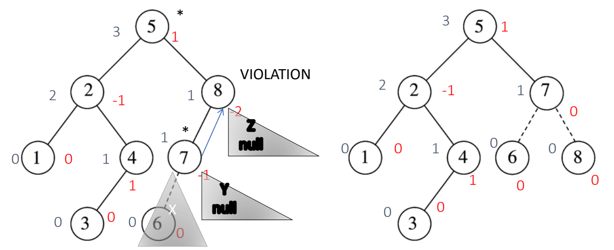
WALK IN OPPOSITE DIRECTION – UPDATE HEIGHTS AND CHECK FOR VIOLATIONS



44

Example, case 1 -> single rotation

WALK IN OPPOSITE DIRECTION – UPDATE HEIGHTS AND CHECK FOR VIOLATIONS

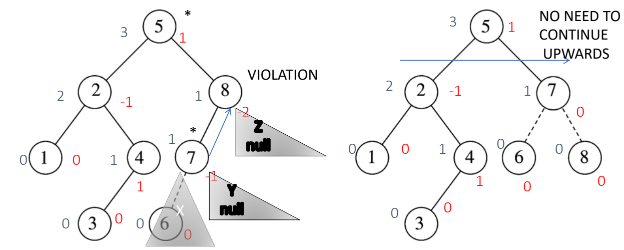


45

45

Example, case 1 -> single rotation

WALK IN OPPOSITE DIRECTION – UPDATE HEIGHTS AND CHECK FOR VIOLATIONS



46

46