AVL TREES

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

1 Introduction

2 Implementing rotations

3 Bibliography

Introduction

AVL: an self-balancing binary search tree

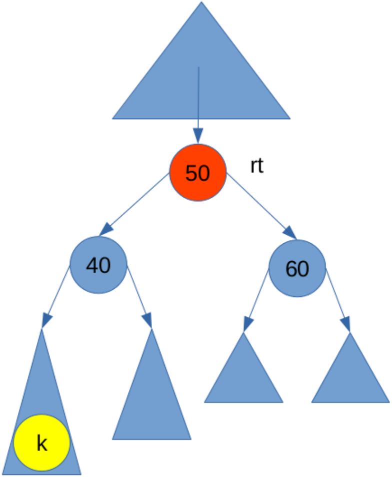
- Proposed in 1962 by G. M. Adelson-Velsky, and E. M. Landis

The balance factor of every node is either -1, 0 or 1

- Difference between the heights of the left and right subtrees
- Height of the empty tree is -1

Rotation: local transformation to rebalance the tree

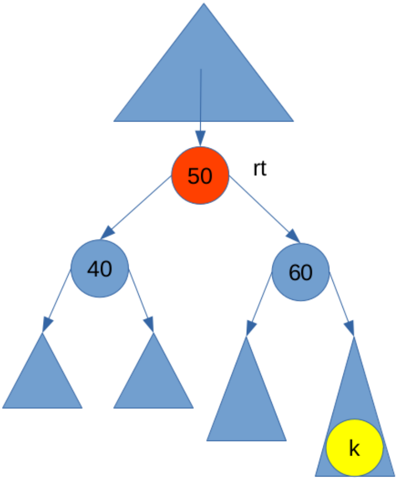
Introduction



R-rotation: key inserted into the left subtree of the left child of *rt*

If $balance > 1 \wedge k < rt.left.key$ then return $rigthRotate(rt)$

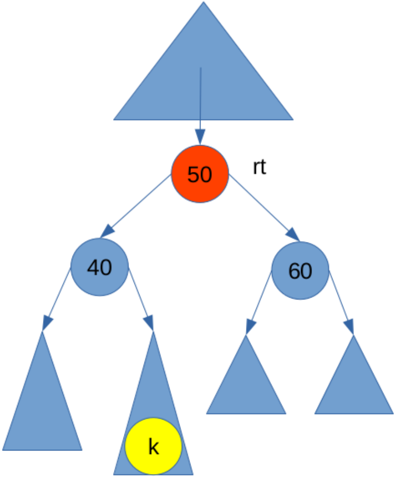
Introduction



L-rotation: key inserted into the right subtree of the right child of *rt*

If $balance < -1 \land k \geq rt.right.key$ then return $leftRotate(rt)$

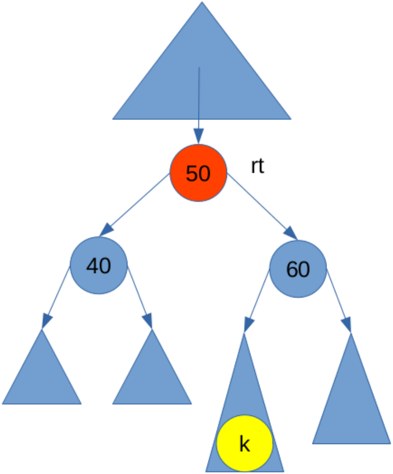
Introduction



LR-rotation: key inserted into the right subtree of the left child of *rt*

If $balance > 1 \wedge k \geq rt.left.key$
then $rt.left \leftarrow leftRotate(rt.left)$,
and return $rigthRotate(rt)$

Introduction



RL-rotation: key inserted into the left subtree of the right child of *rt*

If $balance < -1 \wedge k < rt.right.key$ then $rt.right \leftarrow rightRotate(rt.right)$, and return $leftRotate(rt)$

Agenda

1 Introduction

2 Implementing rotations

3 Bibliography

Implementing rotations

Algorithm: BSTNode inserthelp(BSTNode rt, Key k, E e)

```
if $rt = NULL$ then return $create\_bstnode(k, e)$ ;
if $rt.key > k$ then
    $rt.left \leftarrow inserthelp(rt.left, k, e)$;
else
    $rt.right \leftarrow inserthelp(rt.right, k, e)$;
$rt.height \leftarrow 1 + max(h(rt.left), h(rt.right))$;
int $balance \leftarrow getBalance(rt)$;
if $balance < -1 \wedge k \geq rt.right.key$ then return $leftRotate(rt)$ ;
if $balance > 1 \wedge k < rt.left.key$ then return $rigthRotate(rt)$ ;
if $balance > 1 \wedge k \geq rt.left.key$ then
    $rt.left \leftarrow leftRotate(rt.left)$;
    return $rigthRotate(rt)$;
if $balance < -1 \wedge k < rt.right.key$ then
    $rt.right \leftarrow rightRotate(rt.right)$;
    return $leftRotate(rt)$;
return $rt$;
```

Algorithm: BSTNode inserthelp(BSTNode rt, Key k, E e)

```
if $rt = NULL$ then return $create\_bstnode(k, e)$ ;
if $rt.key > k$ then
    $rt.left \leftarrow inserthelp(rt.left, k, e)$;
else
    $rt.right \leftarrow inserthelp(rt.right, k, e)$;
return $rt$;
```

Implementing rotations

Algorithm: int getBalance(BSTNode rt)

```
if $rt = NULL$ then return 0 ;
return $h(rt.left) - h(rt.right)$;
```

Algorithm: int h(BSTNode rt)

```
if $rt = NULL$ then return $-1$ ;
return $rt.height$;
```

Implementing rotations

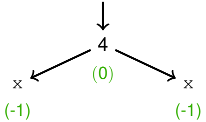
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

Implementing rotations

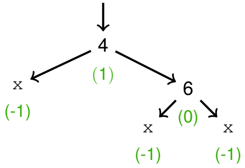
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

Implementing rotations

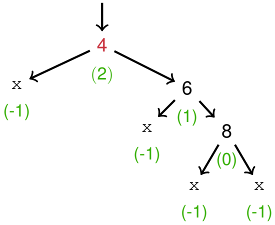
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

4: unbalanced = -1 - 1 = -2

L-rotation: right + right

return *leftRotate*(*rt*)

Implementing rotations

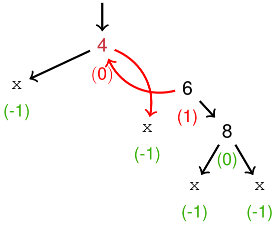
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

4: unbalanced = -1 - 1 = -2

L-rotation: right + right

return $leftRotate(rt)$

Implementing rotations

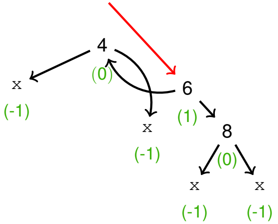
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

after $bst.root \leftarrow inserthelp(\_, \_, \_)$

Implementing rotations

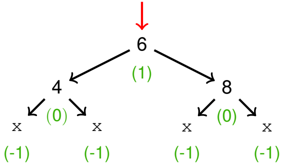
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

better presented as above

Implementing rotations

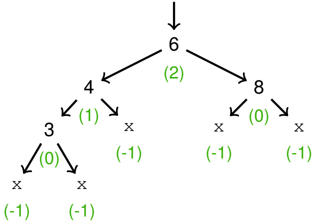
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

Implementing rotations

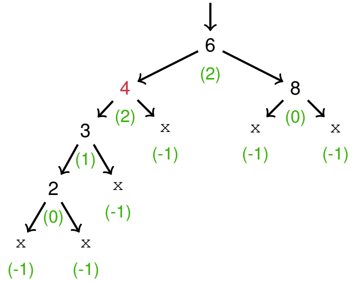
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

4: unbalanced = 1 - -1 = 2

R-rotation: left + left

return *rightRotate*(*rt*)

Implementing rotations

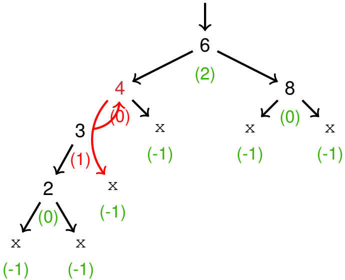
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

4: unbalanced = 1 - -1 = 2

R-rotation: left + left

return $rightRotate(rt)$

Implementing rotations

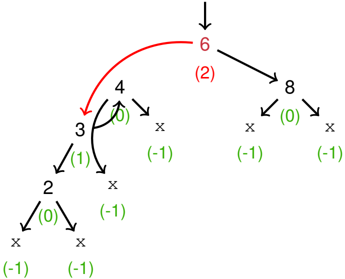
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

after $rt.left \leftarrow inserthelp(\_, \_, \_)$ and $rt.height \leftarrow ...$

Implementing rotations

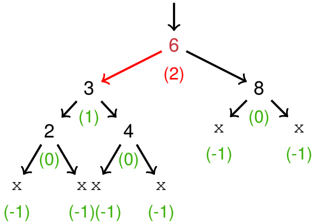
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

better presented as above

Implementing rotations

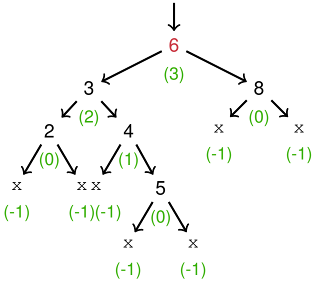
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

6: unbalanced = 2 - 0 = 2

LR-rotation: left + right

$rt.left \leftarrow leftRotate(rt.left)$,

and return $rigthRotate(rt)$

Implementing rotations

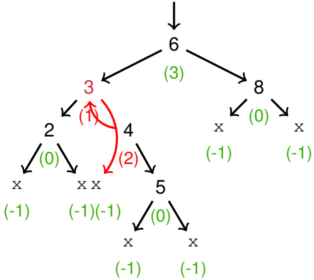
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

6: unbalanced = 2 - 0 = 2

LR-rotation: left + right

$rt.left \leftarrow leftRotate(rt.left)$,

and return $rigthRotate(rt)$

Implementing rotations

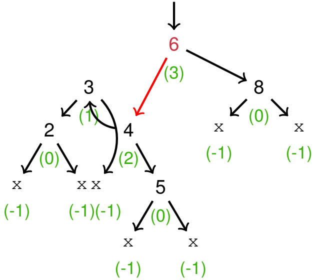
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

6: unbalanced = 2 - 0 = 2

LR-rotation: left + right

$rt.left \leftarrow leftRotate(rt.left)$,

and return $rigthRotate(rt)$

Implementing rotations

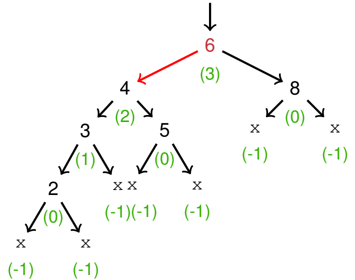
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

6: unbalanced = 2 - 0 = 2

LR-rotation: left + right

$rt.left \leftarrow leftRotate(rt.left)$,

and return $rigthRotate(rt)$

Implementing rotations

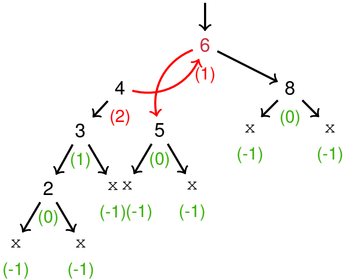
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

6: unbalanced = 2 - 0 = 2

LR-rotation: left + right

$rt.left \leftarrow leftRotate(rt.left)$,

and return $rigthRotate(rt)$

Implementing rotations

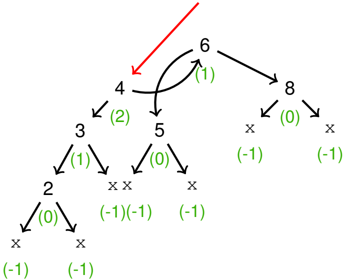
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

after $bst.root \leftarrow inserthelp(\_, \_, \_)$

Implementing rotations

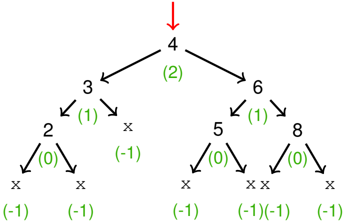
Algorithm: BSTNode rightRotate(BSTNode rt)

```
BSTNode $l \leftarrow rt.left$;
BSTNode $lr \leftarrow l.right$;
$l.right \leftarrow rt$;
$rt.left \leftarrow lr$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$l.height \leftarrow$
  $max(h(l.left), h(l.right)) + 1$;
return $l$;
```

Algorithm: BSTNode leftRotate(BSTNode rt)

```
BSTNode $r \leftarrow rt.right$;
BSTNode $rl \leftarrow r.left$;
$r.left \leftarrow rt$;
$rt.right \leftarrow rl$;
$rt.height \leftarrow$
  $max(h(rt.left), h(rt.right)) + 1$;
$r.height \leftarrow$
  $max(h(r.left), h(r.right)) + 1$;
return $r$;
```

Inserting: 4, 6, 8, 3, 2, 5



Green number: height

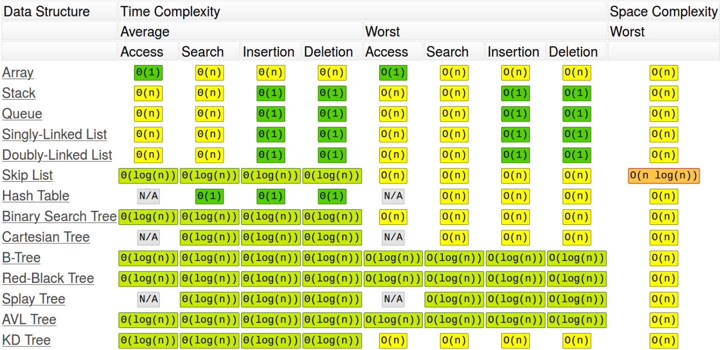
better presented as above

Implementing rotations

Implementing deletion is analogous to the insertion implementation

- Deletion code, in addition to code of rotations
- Updating heights (and balance factors) from the actually removed node (*deletemin*) to the root

Asymptotic efficiency of AVLs[2]



Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

[1]

AVL $\approx$ the same number of comparisons of binary search

- Cons: frequent rotations + storing the node height

[1] http://bigocheatsheet.com/

[2] Source: http://bigocheatsheet.com/

Agenda

1 Introduction

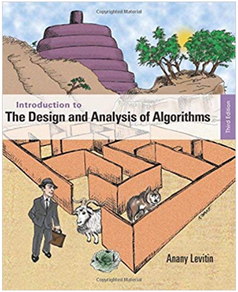
2 Implementing rotations

3 Bibliography

Centro de Informática UFPE

Universidade Federal de Pernambuco

Bibliography



Chapter 6 (pp. 218–223)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3rd edition. Pearson. 2011.

Chapter 13 (pp. 435–437)
Clifford Shaffer.
Data Structures and Algorithm Analysis. Dover, 2013.

AVL TREES

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil