Splay Trees

- invented by Daniel Sleator and Robert Tarjan in 1985

A *splay tree* is a BST + a restructuring operation:

- after each find/insert/remove, that node (or its parent) is brought to the root through *splaying*

Observation.

- frequently-accessed nodes are near the root

Does this ensure O(log n) height?

- on average, yes
- worst case is O(n) – but the worst case is unlikely

Splaying

- *x* is the node being splayed
- *p* is the parent of *x*
- *g* is the parent of *p* (i.e. the grandparent of *x*)

Case 1: zig – applies when *p* is the root

- x is rotated to the root

(the right-handed case is similar)

http://en.wikipedia.org/wiki/Splay_tree

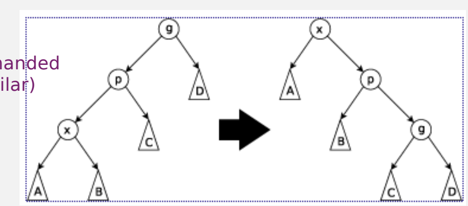
Splaying

- *x* is the node being splayed
- *p* is the parent of *x*
- *g* is the parent of *p* (i.e. the grandparent of *x*)

Case 2: zig-zig – applies when *p* is not the root, and *x* and *p* are both either right children or left children

- p is rotated into g's position, then x is rotated into p's position

(the right-handed case is similar)



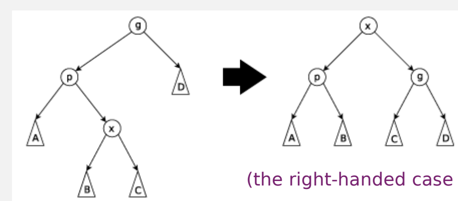
http://en.wikipedia.org/wiki/Splay_tree

Splaying

- *x* is the node being splayed
- *p* is the parent of *x*
- *g* is the parent of *p* (i.e. the grandparent of *x*)

Case 3: zig-zag – applies when *p* is not the root, and one of *x* and *p* is a right child and the other is a left child

- x is rotated into p's position, then x is rotated into g's position



(the right-handed case is similar)

http://en.wikipedia.org/wiki/Splay_tree

Performance

- all operations are O(height) to perform the operation + O(height) splay steps
  - each zig-zig or zig-zag raises *x* two levels, each zig (done at most one per splay) raises *x* one level

  → O(log n) amortized

- worst-case performance
  - splay trees perform as well as optimum static balanced BSTs on sequences of at least *n* accesses (up to a constant factor)
    - "static" = no restructuring of tree after construction
    - "optimal" = tree providing smallest possible time for a series of accesses
  - it is conjectured that splay trees perform as well as optimum dynamic balanced BSTs on sequences of at least *n* accesses (up to a constant factor)
    - "dynamic" = tree can be restructured after construction (e.g. AVL trees, red-black trees)