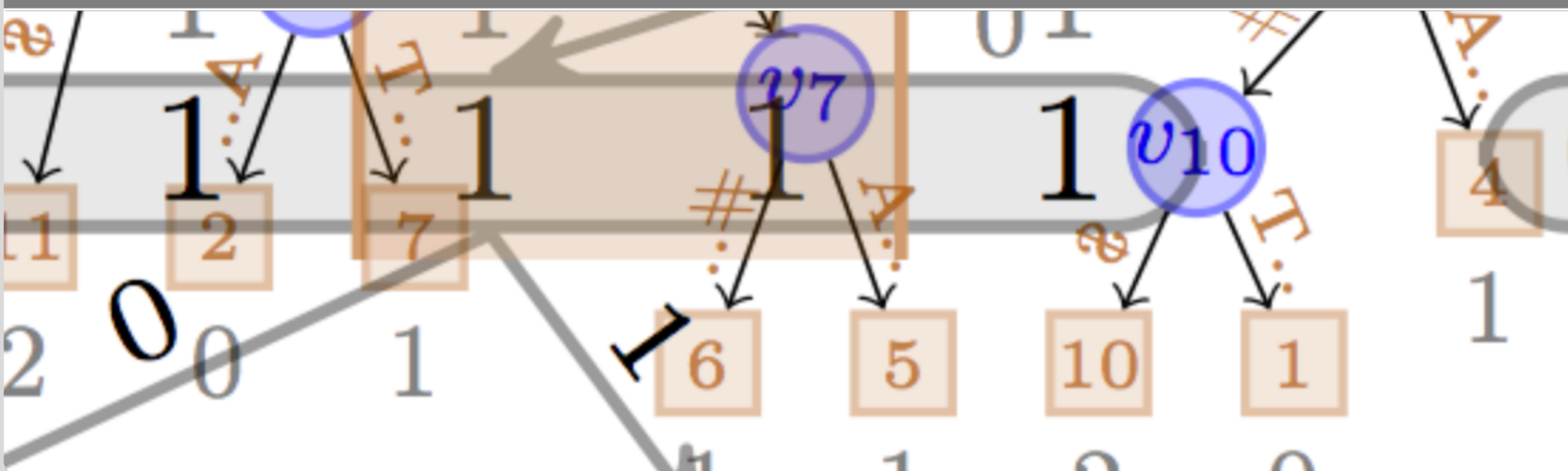
Advanced Data Structures: Splay Trees

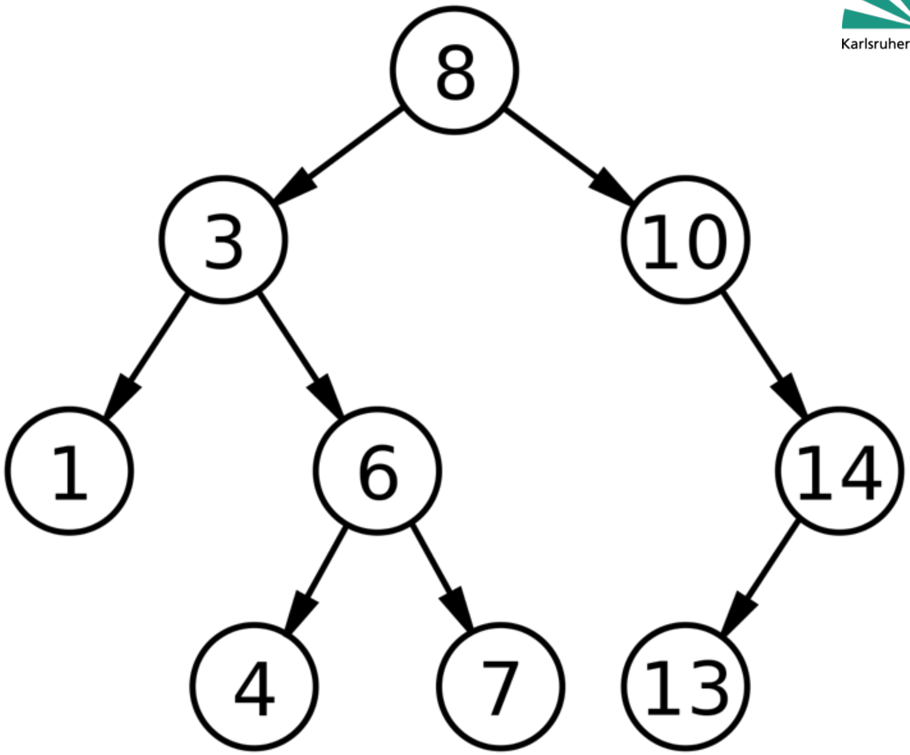
Samuel Groß

Institute of Theoretical Informatics - Algorithmics



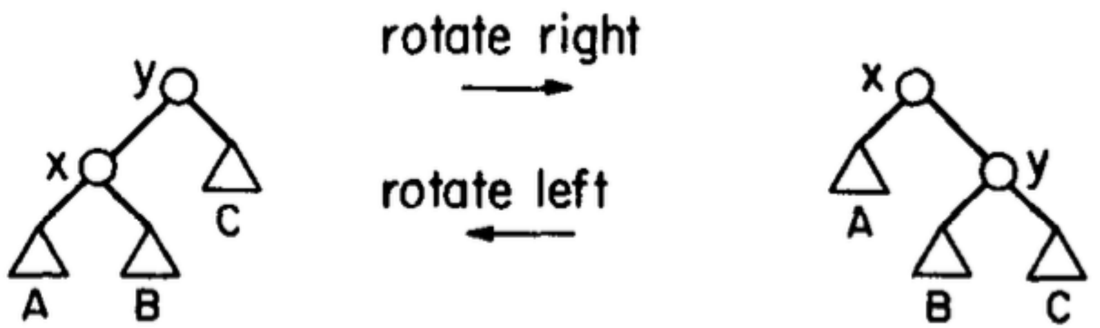
Context

- Binary search tree
- Goal: cheap
  - Access
  - Insert
  - Delete
- Self-adjustment
- Amortized analysis (study sequences of operations)
- Invented in 1985 [1]



Prerequisites

- Definitions:
  - Parent node: $p(x)$
  - Grandparent node: $g(x) = p(p(x))$
- Operations:
  - Node rotations



Inspiration: Move to Root

Karlsruher Institut für Technologie

- Restructure tree after/during operations
- Simple heuristic: move accessed node to root through multiple rotations

Inspiration: Move to Root

- Restructure tree after/during operations
- Simple heuristic: move accessed node to root through multiple rotations

- But: doesn’t help with linear chains

==> Need something better: splaying!

Splay Operation

Karlsruher Institut für Technologie

- Also based on simple rotations
- Splay operation considers path from node to grandparent
- 3 Cases: *zig*, *zig-zig*, *zig-zag*
- Repeat splaying until accessed node is root of the tree

Zig

- Situation: *p(x)* is the root
  - `rotate(x, p(x))`

Zig-Zig

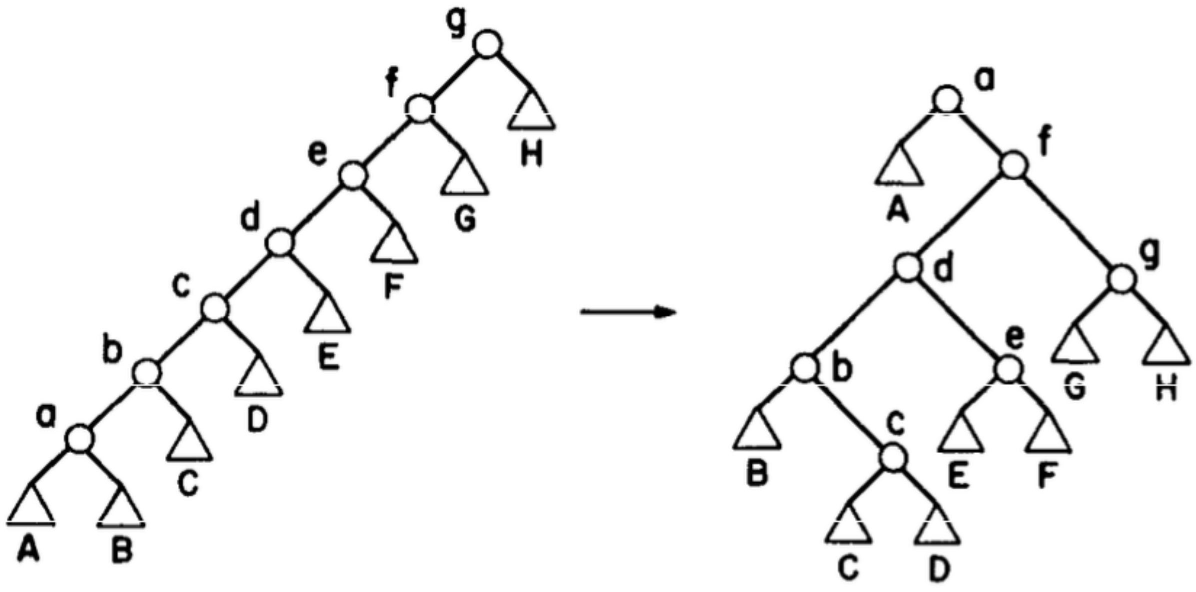
- Situation: *x* and *p(x)* are both left (right) children
  - `rotate(p(x), g(x))`
  - `rotate(x, p(x))`

Zig-Zag

- Situation: *x* left child, *p(x)* right child (or vice versa)
  - `rotate(x, p(x))`
  - `rotate(x, p(x))`

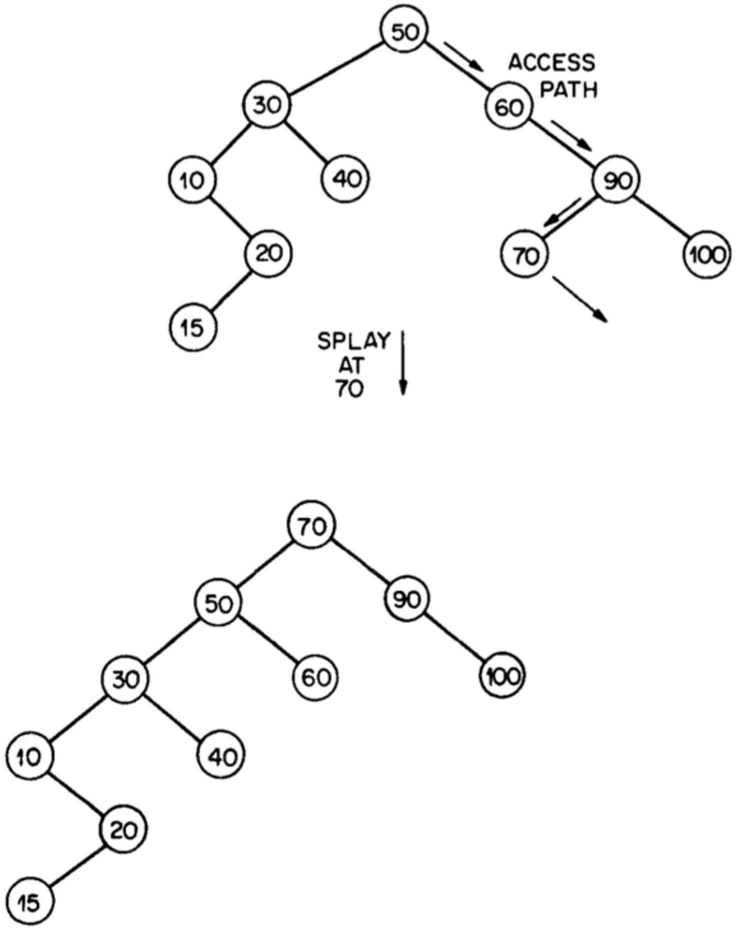
Benefits

- Average height decreases



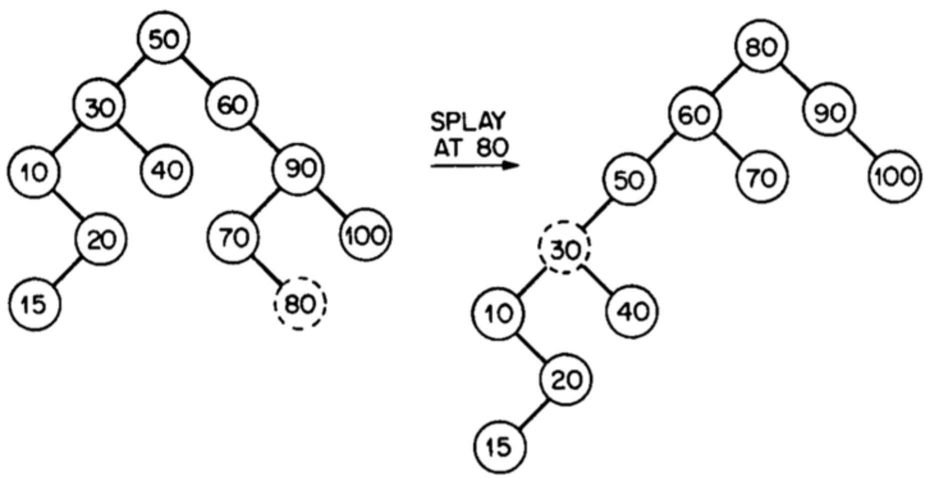
Operation Access

- BST traversal until node is found (or null pointer)
- Splay at node *x* (or its parent if *x* not found)
- In this example: `access(80)`



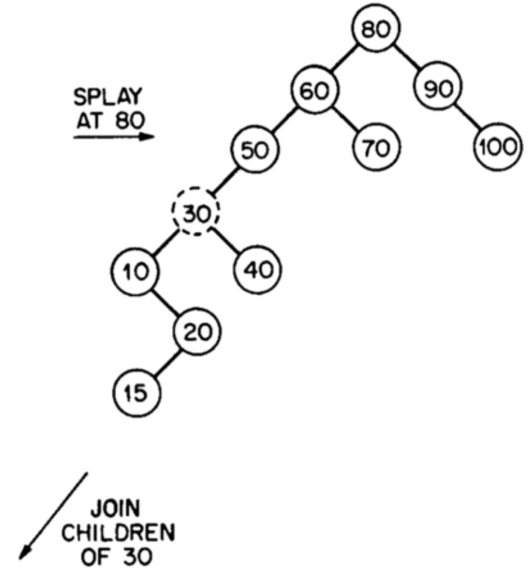
Operation Insert

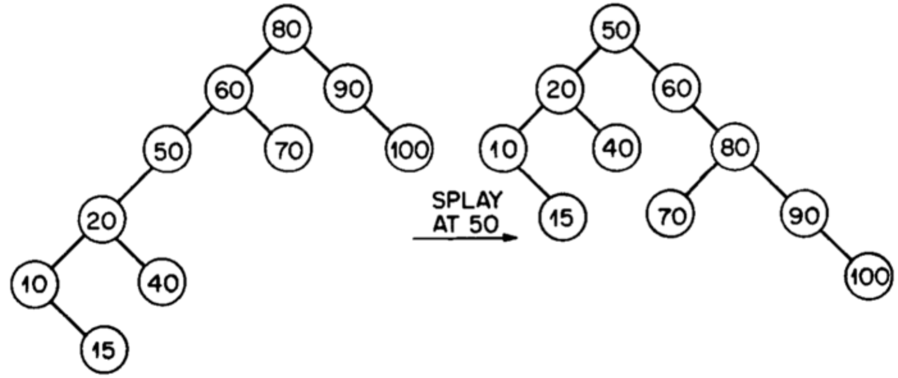
- BST traversal until parent is found
- Insert new node
- Splay at new node
- Example: `insert(80)`



Operation Delete

- Join subtrees of node *x*
  - Splay at largest node in left subtree
  - Add right subtree as right child of new root
- Replace *x* with joined subtree
- Splay at *p(x)*
- Example: `delete(30)`





Proof

Amortized Analysis: Potential Method

- Idea: assign *potential* (scalar) to state of tree
  - Roughly equivalent to “unbalancedness" of tree
- amortized cost = real cost + difference in potential:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

- Then:

$$\sum_{i=0}^{m} a_i = \sum_{i=0}^{m} t_i + \Phi_m - \Phi_0 \Leftrightarrow \sum_{i=0}^{m} t_i = \sum_{i=0}^{m} a_i + \Phi_0 - \Phi_m$$

$$\Leftrightarrow t = a + \Phi_0 - \Phi_m$$

Potential

Find upper bound for amortized cost *and* the total amount of change in the tree's potential of a sequence of operations

==> yields upper bound for the actual time

$$t = a + \Phi_0 - \Phi_m$$

$$a \leq X \qquad\qquad \Phi_0 - \Phi_m \leq Y$$

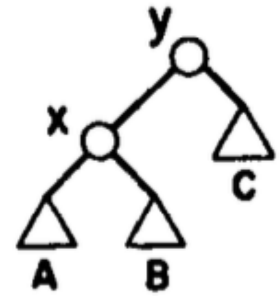
$$\Rightarrow t \leq X + Y$$

Potential of a Splay Tree (simplified)

- Definitions
  - $s(x)$: number of nodes in tree rooted at x (size)
  - $r(x)$: $log2(s(x))$ (rank)
  - $potential(t)$: sum over all ranks of the nodes of t

Access Lemma: armortized time to splay a tree with root t at node x is at most $3(r(t) - r(x)) + 1 = O(log(s(t)/s(x)))$ $[= O(log(n))]$

Zig Case



Single rotation, so amortized cost is

$$1 + r'(x) + r'(y) - r(x) - r(y)$$

since only *x* and *y* can change rank

$$\leq 1 + r'(x) - r(x)$$

since *r(y) >= r'(y)*

$$\leq 1 + 3(r'(x) - r(x))$$

since *r'(x) >= r(x)*

Zig-Zig

Two rotations, so amortized cost is

$$2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$

$$= \ldots \leq 2 + r'(x) + r'(z) - 2r(x)$$

Claim: $2 + r'(x) + r'(z) - 2r(x) \leq 3(r'(x) - r(x))$

$$\Leftrightarrow \ldots \Leftrightarrow \log\left(s(x)/s'(x)\right) + \log\left(s'(z)/s'(x)\right) \leq -2$$

True since $max(\log x + \log y,\ \ x, y > 0, x + y \leq 1) = -2$

And $s(x) + s'(z) \leq s'(x)$

Finishing

- Found upper bound for amortized cost for *m* operations
- Need upper bound for maximum change in potential:
  - Maximum rank: *log(n)*
  - Maximum potential: *n log(n)*
  - Maximum decrease in potential: *n log(n)*
- Thus, total cost of *m* splay operations:

$$O(m \log n + n \log n)$$

Evaluation

- Pros:
  - Cheap operations
  - No additional metadata required
  - Efficient implementation possible
- Cons:
  - “Expensive” lookup, due to splaying
  - No real-time guarantees

References

[1] Sleator, Daniel Dominic, and Robert Endre Tarjan. "Self-adjusting binary search trees." *Journal of the ACM (JACM)* 32.3 (1985): 652-686.

Potential of a Splay Tree

- More general: assign arbitrary weight to every node
- Definitions
  - weight(x)/w(x): weight of node x
  - size(x)/s(x): sum of the weights of all nodes in subtree
  - rank(x)/r(x): log2(size(x))
  - potential(t): sum over all ranks of the nodes of t
- Similar amortized analysis, but further results possible through choice of weights
- Previous analysis is special case with w(x) = 1 for every node