3-2 Amortized Analysis

(Part II: In-Depth Examples)

Hengfeng Wei

hfwei@nju.edu.cn

October 15, 2018

Robert Tarjan

SIAM J. ALG. DISC. METH.
Vol. 6, No. 2, April 1985

AMORTIZED COMPUTATIONAL COMPLEXITY*

ROBERT ENDRE TARJAN†

Abstract. A powerful technique in the complexity analysis of data structures is *amortization*, or averaging over time. Amortized running time is a realistic but robust complexity measure for which we can obtain surprisingly tight upper and lower bounds on a variety of algorithms. By following the principle of designing algorithms whose amortized complexity is low, we obtain "self-adjusting" data structures that are simple, flexible and efficient. This paper surveys recent work by several researchers on amortized complexity.

"Amortized Computational Complexity", 1985

The Potential Method

$$D_0,\ o_1,\ D_1,\ o_2,\ \cdots,\ \underbrace{D_{i-1},\ o_i,\ D_i}_{\text{the } i\text{-th operation}},\ \cdots,\ D_{n-1},\ o_n,\ D_n$$

$$D_0,\ o_1,\ D_1,\ o_2,\ \cdots,\ \underbrace{D_{i-1},\ o_i,\ D_i}_{\text{the } i\text{-th operation}},\ \cdots,\ D_{n-1},\ o_n,\ D_n$$

$$\Phi : \left\{ D_i \mid 0 \le i \le n \right\} \to \mathcal{R}$$

$$D_0,\ o_1,\ D_1,\ o_2,\ \cdots,\ \underbrace{D_{i-1},\ o_i,\ D_i}_{\text{the } i\text{-th operation}},\ \cdots,\ D_{n-1},\ o_n,\ D_n$$

$$\Phi : \Big\{ D_i \mid 0 \le i \le n \Big\} \to \mathcal{R}$$

$$\boxed{\hat{c}_i = c_i + \Big(\Phi(D_i) - \Phi(D_{i-1}) \Big)}$$

$$D_0,\ o_1,\ D_1,\ o_2,\ \cdots,\ \underbrace{D_{i-1},\ o_i,\ D_i}_{\text{the } i\text{-th operation}},\ \cdots,\ D_{n-1},\ o_n,\ D_n$$

$$\Phi : \Big\{ D_i \mid 0 \le i \le n \Big\} \to \mathcal{R}$$

$$\boxed{\hat{c}_i = c_i + \Big(\Phi(D_i) - \Phi(D_{i-1}) \Big)}$$

$$\sum_{1 \le i \le n} c_i = \left(\sum_{1 \le i \le n} \hat{c}_i \right) + \Big(\underbrace{\Phi(D_0) - \Phi(D_n)}_{\text{net decrease in potential}} \Big)$$

$$\sum_{1 \le i \le n} c_i = \left(\sum_{1 \le i \le n} \hat{c}_i \right) + \Big(\underbrace{\Phi(D_0) - \Phi(D_n)}_{\text{net decrease in potential}} \Big)$$

$$\sum_{1 \le i \le n} c_i = \left(\sum_{1 \le i \le n} \hat{c}_i \right) + \Big(\underbrace{\Phi(D_0) - \Phi(D_n)}_{\text{net decrease in potential}} \Big)$$

$$\underbrace{\Phi(D_0) - \Phi(D_n)}_{\text{net decrease in potential}} \le \square \implies \boxed{\sum_{1 \le i \le n} c_i \le \left(\sum_{1 \le i \le n} \hat{c}_i \right) + \square}$$

$$\boxed{\sum_{1 \le i \le n} c_i \le \left(\sum_{1 \le i \le n} \hat{c}_i \right) + \square}$$

$$\boxed{\sum_{1 \le i \le n} c_i \le \left(\sum_{1 \le i \le n} \hat{c}_i \right) + \square}$$

$$\square = 0 \ (\forall i,\ \Phi(D_i) \ge \Phi(D_0)) \implies \forall n,\ \sum_{1 \le i \le n} c_i \le \sum_{1 \le i \le n} \hat{c}_i$$

$$\boxed{\sum_{1 \le i \le n} c_i \le \left(\sum_{1 \le i \le n} \hat{c}_i \right) + \square}$$

$$\square = 0 \ (\forall i,\ \Phi(D_i) \ge \Phi(D_0)) \implies \forall n,\ \sum_{1 \le i \le n} c_i \le \sum_{1 \le i \le n} \hat{c}_i$$

$$\Phi(D_0) = 0$$

$$\forall 1 \le i \le n :\ \Phi(D_i) \ge 0$$

What work are you proudest of?

What work are you proudest of?

Proudest? It's hard to choose.

What work are you proudest of?

Proudest? It's hard to choose.

I like the ***self-adjusting search tree*** *data structure*

that Danny Sleator and I developed.

Self-Adjusting Binary Search Trees

DANIEL DOMINIC SLEATOR AND ROBERT ENDRE TARJAN

AT&T Bell Laboratories, Murray Hill, NJ

Abstract. The *splay* tree, a self-adjusting form of binary search tree, is developed and analyzed. The binary search tree is a data structure for representing tables and lists so that accessing, inserting, and deleting items is easy. On an n-node splay tree, all the standard search tree operations have an amortized time bound of $O(\log n)$ per operation, where by "amortized time" is meant the time per operation averaged over a worst-case sequence of operations. Thus splay trees are as efficient as balanced trees when total running time is the measure of interest. In addition, for sufficiently long access sequences, splay trees are as efficient, to within a constant factor, as static optimum search trees. The efficiency of splay trees comes not from an explicit structural constraint, as with balanced trees, but from applying a simple restructuring heuristic, called *splaying*, whenever the tree is accessed. Extensions of splaying give simplified forms of two other data structures: lexicographic or multidimensional search trees and link/cut trees.

"Self-Adjusting Binary Search Trees – Splay Tree", JACM, 1985

Self-Adjusting Binary Search Trees

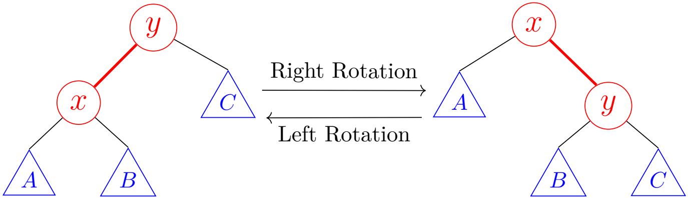
vs. Balanced Binary Search Trees

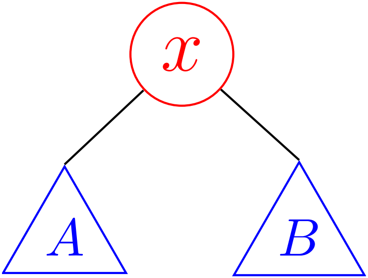
$\textsc{Splay}(x)$:

Moving node x to the root of the tree by performing a sequence of rotations along the path from x to the root.

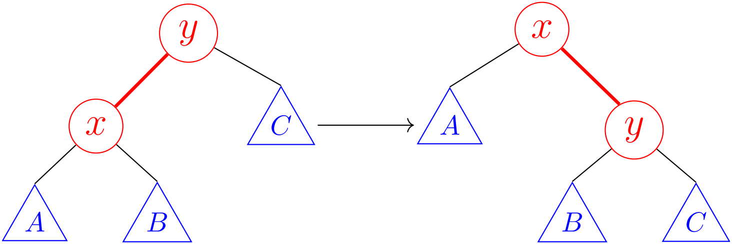
$\textsc{Splay}(x)$:

Moving node x to the root of the tree by performing a sequence of rotations along the path from x to the root.



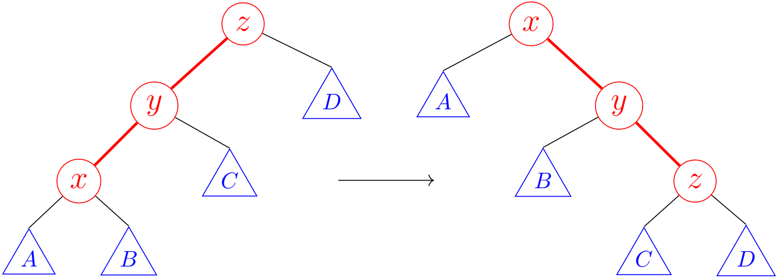


CASE 0: x is the root



CASE 1: zig

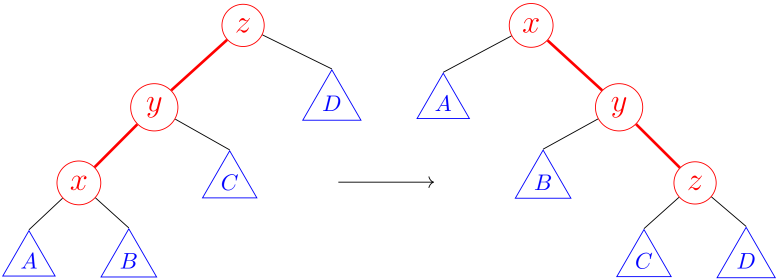
$y = p(x)$ is the root



CASE 2: zig-zig

$$y = p(x) \qquad z = p(y)$$

$$x = lc(y) \qquad y = lc(z)$$

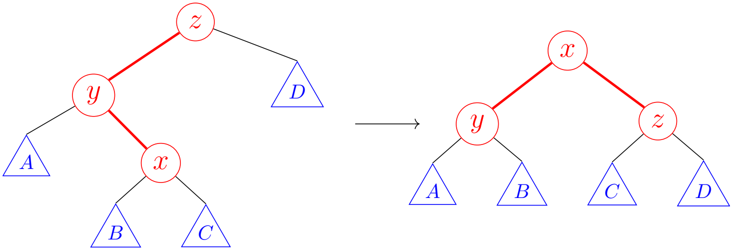


Case 2: zig-zig

$$y = p(x) \qquad z = p(y)$$

$$x = lc(y) \qquad y = lc(z)$$

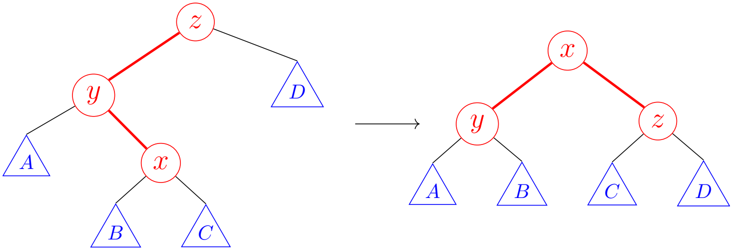
$$(1): y - z \qquad (2): x - y$$



CASE 3: zig-zag

$$y = p(x) \qquad z = p(y)$$

$$x = rc(y) \qquad y = lc(z)$$

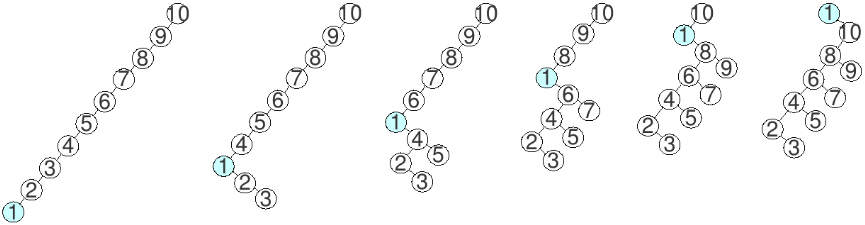


Case 3: zig-zag

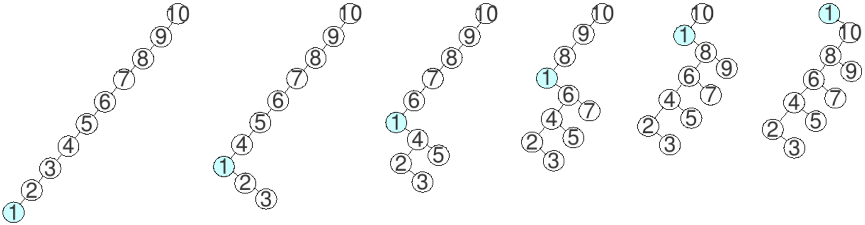
$$y = p(x) \qquad z = p(y)$$

$$x = rc(y) \qquad y = lc(z)$$

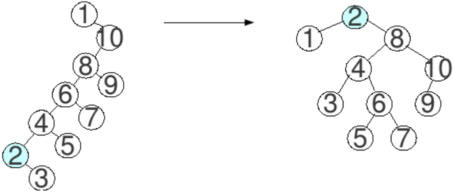
$$(1) : x - y \qquad (2) : x - z$$



Splay(1)



SPLAY(1)



SPLAY(2)

Amortized analysis of SPLAY

Amortized analysis of SPLAY

A splay tree T of n-node

An arbitrary sequence of m SPLAY operations

Amortized analysis of SPLAY

A splay tree T of n-node

An arbitrary sequence of m SPLAY operations

$\#$ of rotations

Amortized analysis of Splay

A splay tree T of n-node

An arbitrary sequence of m Splay operations

$\#$ of rotations

Theorem

$$\hat{c}_{\text{Splay}} = O(\log n).$$

$$\Phi_0 \ \textsc{Splay}_1 \ \Phi_1 \ \textsc{Splay}_2 \ \Phi_2 \ \cdots \underbrace{\Phi_{i-1} \ \textsc{Splay}_i \ \Phi_i}_{\text{the } i\text{-th } \textsc{Splay}} \cdots \textsc{Splay}_m \ \Phi_m$$

$$\hat{c}_{\textsc{Splay}_i} = c_{\textsc{Splay}_i} + \left(\Phi_{\textsc{Splay}_i} - \Phi_{\textsc{Splay}_{i-1}}\right)$$

$$\Phi_0 \;\; \textsc{Splay}_1 \;\; \Phi_1 \;\; \textsc{Splay}_2 \;\; \Phi_2 \; \cdots \; \underbrace{\Phi_{i-1} \;\; \textsc{Splay}_i \;\; \Phi_i}_{\text{the } i\text{-th } \textsc{Splay}} \cdots \; \textsc{Splay}_m \;\; \Phi_m$$

$$\hat{c}_{\textsc{Splay}_i} = c_{\textsc{Splay}_i} + \left(\Phi_{\textsc{Splay}_i} - \Phi_{\textsc{Splay}_{i-1}}\right)$$

How to define Φ?

$s(x)$: # of nodes in the subtree rooted at x

$s(x)$: # of nodes in the subtree rooted at x

$$r(x) = \log s(x)$$

$s(x)$: # of nodes in the subtree rooted at x

$$r(x) = \log s(x)$$

$$\Phi = \sum_{x \in T} r(x)$$

$s(x)$: # of nodes in the subtree rooted at x

$$r(x) = \log s(x)$$

$$\Phi = \sum_{x \in T} r(x)$$

$s(x)$: # of nodes in the subtree rooted at x

$$r(x) = \log s(x)$$

$$\Phi = \sum_{x \in T} r(x)$$

$$\hat{c}_{\text{SPLAY}_i} = c_{\text{SPLAY}_i} + (\Phi_{\text{SPLAY}_i} - \Phi_{\text{SPLAY}_{i-1}})$$

How to calculate $(\Phi_{\text{SPLAY}_i} - \Phi_{\text{SPLAY}_{i-1}})$ *and* c_{SPLAY_i}*?*

$$\Phi_0 \ \textsc{Splay}_1 \ \Phi_1 \ \textsc{Splay}_2 \ \Phi_2 \cdots \underbrace{\Phi_{i-1} \ \textsc{Splay}_i \ \Phi_i}_{\text{the } i\text{-th } \textsc{Splay}} \cdots \textsc{Splay}_m \ \Phi_m$$

$$\Phi_0 \;\; \textsc{Splay}_1 \;\; \Phi_1 \;\; \textsc{Splay}_2 \;\; \Phi_2 \;\cdots\; \underbrace{\Phi_{i-1} \;\; \textsc{Splay}_i \;\; \Phi_i}_{\text{the } i\text{-th } \textsc{Splay}} \cdots\; \textsc{Splay}_m \;\; \Phi_m$$

$$\underbrace{\Phi_{i-1} \;\; \textsc{Splay}_i \;\; \Phi_i}_{\text{the } i\text{-th } \textsc{Splay}} :$$

$$\Phi_{i-1} \;\triangleq\; \Phi_{0'} \; \textsc{Iter}_1 \; \Phi_{1'} \;\cdots\; \underbrace{\Phi_{k-1} \; \textsc{Iter}_k \; \Phi_k}_{\text{the } k\text{-th } \textsc{Iteration}} \cdots\; \textsc{Iter}_l \; \Phi_l \;\triangleq\; \Phi_i$$

$$\Phi_0 \;\; \textsc{Splay}_1 \;\; \Phi_1 \;\; \textsc{Splay}_2 \;\; \Phi_2 \; \cdots \; \underbrace{\Phi_{i-1} \;\; \textsc{Splay}_i \;\; \Phi_i}_{\text{the } i\text{-th } \textsc{Splay}} \cdots \; \textsc{Splay}_m \;\; \Phi_m$$

$$\underbrace{\Phi_{i-1} \;\; \textsc{Splay}_i \;\; \Phi_i}_{\text{the } i\text{-th } \textsc{Splay}} :$$

$$\Phi_{i-1} \;\triangleq\; \Phi_{0'} \; \textsc{Iter}_1 \; \Phi_{1'} \;\; \cdots \;\; \underbrace{\Phi_{k-1} \; \textsc{Iter}_k \; \Phi_k}_{\text{the } k\text{-th } \textsc{Iteration}} \cdots \;\; \textsc{Iter}_l \; \Phi_l \;\triangleq\; \Phi_i$$

$$\begin{aligned} \hat{c}_{\textsc{Splay}_i} &= \sum_{1 \le j \le l} \hat{c}_{\textsc{Iter}_j} \\ &= \sum_{1 \le j \le l} c_{\textsc{Iter}_j} + \left(\Phi_{\textsc{Iter}_j} - \Phi_{\textsc{Iter}_{j-1}}\right) \end{aligned}$$

$$\hat{c}_{\textsc{Iter}_j} = c_{\textsc{Iter}_j} + (\Phi_{\textsc{Iter}_j} - \Phi_{\textsc{Iter}_{j-1}})$$

$$\hat{c}_{\text{ITER}_j} = c_{\text{ITER}_j} + \left(\Phi_{\text{ITER}_j} - \Phi_{\text{ITER}_{j-1}}\right)$$

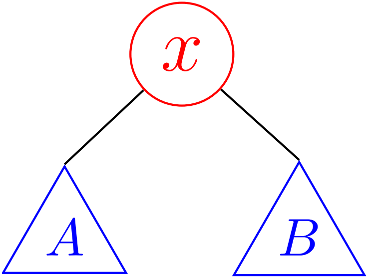
By Case Analysis.

$$\hat{c}_{\textsc{Iter}_j} = c_{\textsc{Iter}_j} + (\Phi_{\textsc{Iter}_j} - \Phi_{\textsc{Iter}_{j-1}})$$

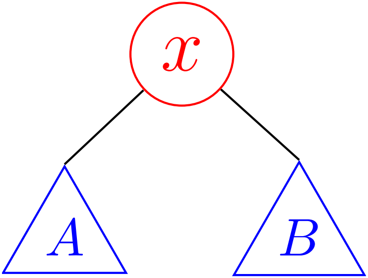
By Case Analysis.

$$\hat{c}_j = c_j + (\Phi_j - \Phi_{j-1})$$

Remember: ITER

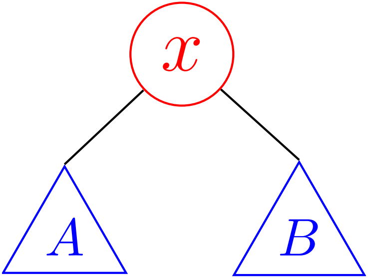


CASE 0



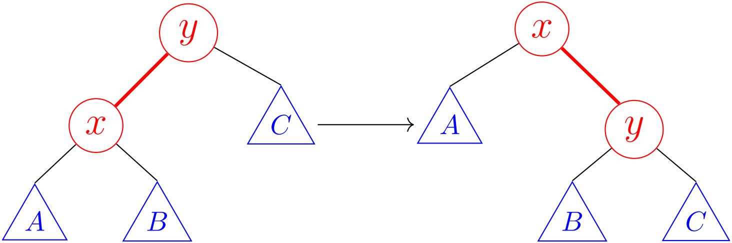
CASE 0

$$\hat{c}_j = c_j + (\Phi_j - \Phi_{j-1})$$



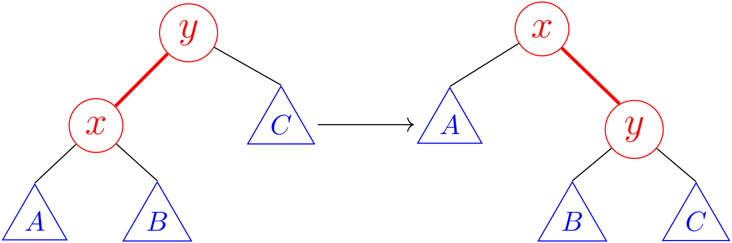
CASE 0

$$\begin{aligned}\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1})\\ &= 0 + 0\\ &= 0\end{aligned}$$



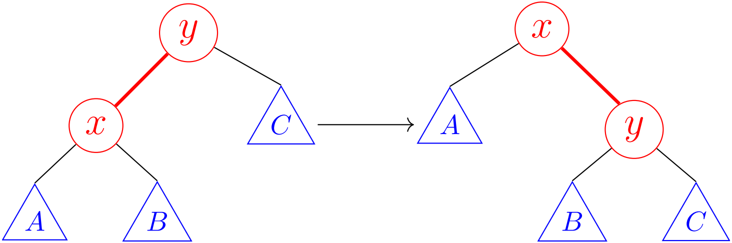
CASE 1: zig

$$\hat{c}_j = c_j + (\Phi_j - \Phi_{j-1})$$



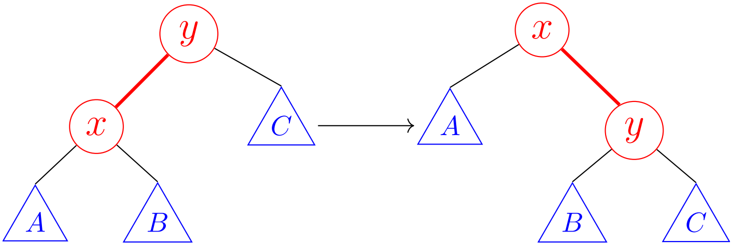
CASE 1: zig

$$
\begin{aligned}
\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1}) \\
&= 1 + r_j(x) + r_j(y) - r_{j-1}(x) - r_{j-1}(y)
\end{aligned}
$$



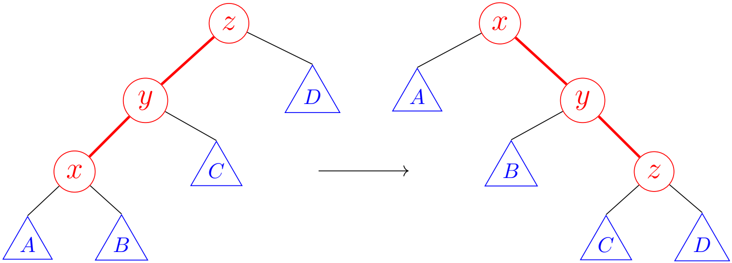
CASE 1: zig

$$
\begin{aligned}
\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1}) \\
&= 1 + r_j(x) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) \\
&\leq 1 + r_j(x) - r_{j-1}(x)
\end{aligned}
$$



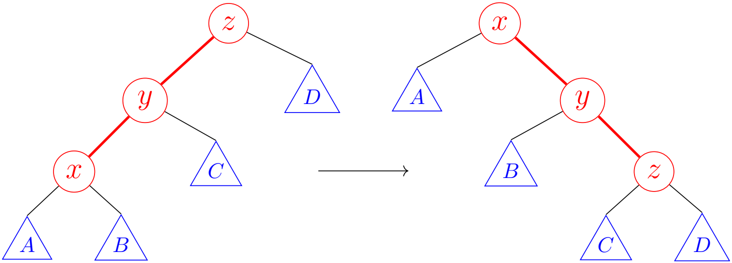
Case 1: zig

$$
\begin{aligned}
\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1}) \\
&= 1 + r_j(x) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) \\
&\leq 1 + r_j(x) - r_{j-1}(x) \\
&\leq 1 + 3\big(r_j(x) - r_{j-1}(x)\big)
\end{aligned}
$$



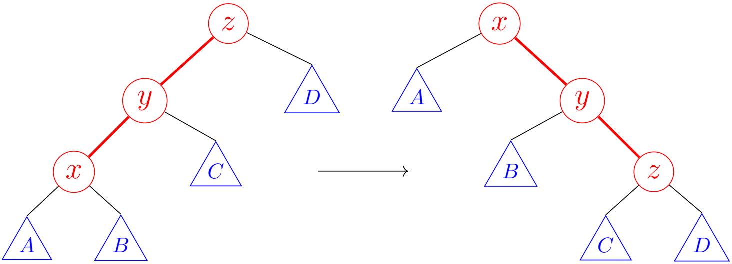
CASE 2: zig-zig

$$\hat{c}_j = c_j + (\Phi_j - \Phi_{j-1})$$



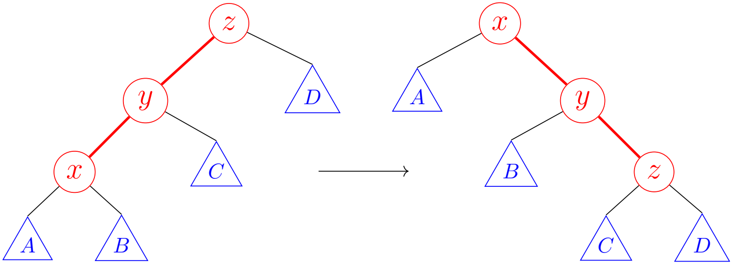
CASE 2: zig-zig

$$
\begin{aligned}
\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1}) \\
&= 2 + r_j(x) + r_j(y) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) - r_{j-1}(z)
\end{aligned}
$$



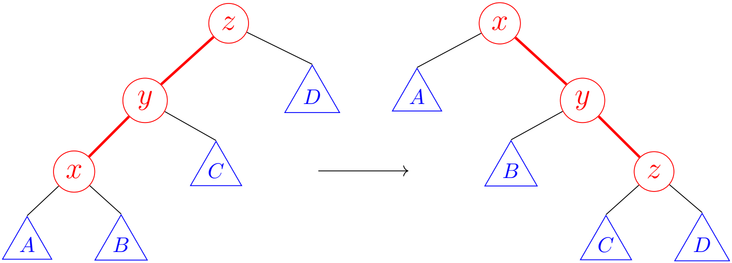
CASE 2: zig-zig

$$\begin{aligned}\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1}) \\ &= 2 + r_j(x) + r_j(y) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) - r_{j-1}(z) \\ &= 2 + r_j(y) + r_j(y) - r_{j-1}(x) - r_{j-1}(y)\end{aligned}$$



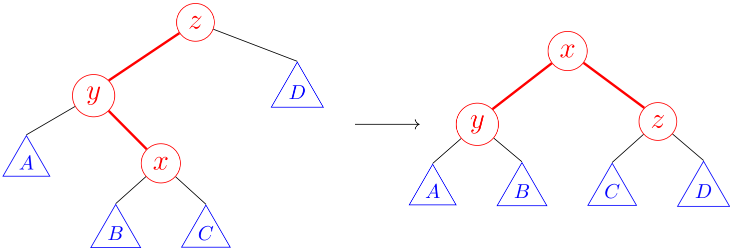
CASE 2: zig-zig

$$
\begin{aligned}
\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1}) \\
&= 2 + r_j(x) + r_j(y) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) - r_{j-1}(z) \\
&= 2 + r_j(y) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) \\
&\leq 2 + r_j(x) + r_j(z) - 2r_{j-1}(x)
\end{aligned}
$$



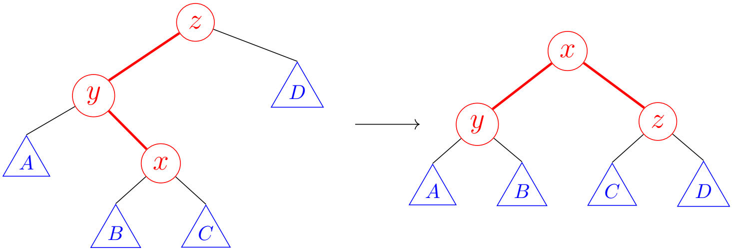
CASE 2: zig-zig

$$
\begin{aligned}
\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1}) \\
&= 2 + r_j(x) + r_j(y) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) - r_{j-1}(z) \\
&= 2 + r_j(y) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) \\
&\le 2 + r_j(x) + r_j(z) - 2r_{j-1}(x) \\
&\le 3\big(r_j(x) - r_{j-1}(x)\big)
\end{aligned}
$$



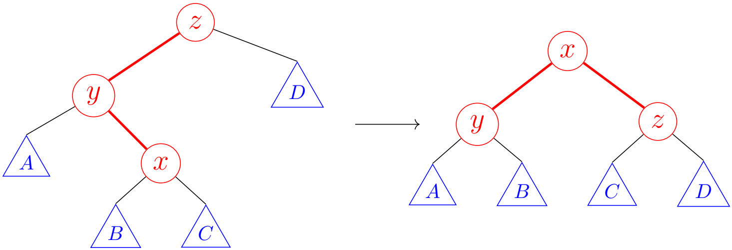
CASE 3: zig-zag

$$\hat{c}_j = c_j + (\Phi_j - \Phi_{j-1})$$



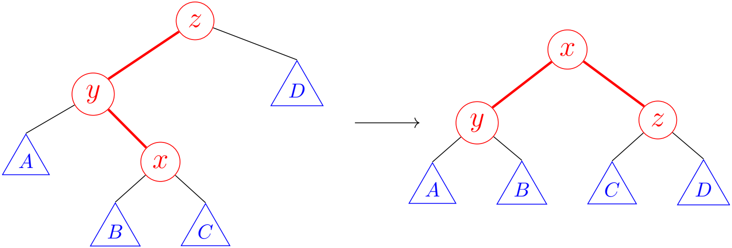
CASE 3: zig-zag

$$
\begin{aligned}
\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1}) \\
&= 2 + r_j(x) + r_j(y) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) - r_{j-1}(z)
\end{aligned}
$$



CASE 3: zig-zag

$$
\begin{aligned}
\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1}) \\
&= 2 + r_j(x) + r_j(y) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) - r_{j-1}(z) \\
&\leq 2 + r_j(y) + r_j(z) - 2r_{j-1}(x)
\end{aligned}
$$



CASE 3: zig-zag

$$
\begin{aligned}
\hat{c}_j &= c_j + (\Phi_j - \Phi_{j-1}) \\
&= 2 + r_j(x) + r_j(y) + r_j(y) - r_{j-1}(x) - r_{j-1}(y) - r_{j-1}(z) \\
&\leq 2 + r_j(y) + r_j(z) - 2r_{j-1}(x) \\
&\leq 3\big(r_j(x) - r_{j-1}(x)\big)
\end{aligned}
$$

$$\hat{c}_{\textsc{Iter}_j} \le \begin{cases} 0, & \textsc{Case } 0 \\ 1 + 3\big(r_j(x) - r_{j-1}(x)\big), & \textsc{Case } 1 \\ 3\big(r_j(x) - r_{j-1}(x)\big), & \textsc{Case } 2 \\ 3\big(r_j(x) - r_{j-1}(x)\big), & \textsc{Case } 3 \end{cases}$$

$$\hat{c}_{\text{ITER}_j} \leq \begin{cases} 0, & \textsc{Case } 0 \\ 1 + 3\big(r_j(x) - r_{j-1}(x)\big), & \textsc{Case } 1 \\ 3\big(r_j(x) - r_{j-1}(x)\big), & \textsc{Case } 2 \\ 3\big(r_j(x) - r_{j-1}(x)\big), & \textsc{Case } 3 \end{cases}$$

$$\begin{aligned} \hat{c}_{\text{SPLAY}_i} &= \sum_{1 \leq j \leq l} \hat{c}_{\text{ITER}_j} \\ &= \sum_{1 \leq j \leq l} c_{\text{ITER}_j} + \big(\Phi_{\text{ITER}_j} - \Phi_{\text{ITER}_{j-1}}\big) \end{aligned}$$

$$\hat{c}_{\textsc{Iter}_j} \leq \begin{cases} 0, & \textsc{Case } 0 \\ 1 + 3\big(r_j(x) - r_{j-1}(x)\big), & \textsc{Case } 1 \\ 3\big(r_j(x) - r_{j-1}(x)\big), & \textsc{Case } 2 \\ 3\big(r_j(x) - r_{j-1}(x)\big), & \textsc{Case } 3 \end{cases}$$

$$\begin{aligned} \hat{c}_{\textsc{Splay}_i} &= \sum_{1 \leq j \leq l} \hat{c}_{\textsc{Iter}_j} \\ &= \sum_{1 \leq j \leq l} c_{\textsc{Iter}_j} + \left(\Phi_{\textsc{Iter}_j} - \Phi_{\textsc{Iter}_{j-1}}\right) \\ &\leq 3\big(r_{\textsc{Iter}_l}(x) - r_{\textsc{Iter}_0}(x)\big) \; {\color{red}+1} \end{aligned}$$

$$\hat{c}_{\text{ITER}_j} \leq \begin{cases} 0, & \text{CASE } 0 \\ 1 + 3\big(r_j(x) - r_{j-1}(x)\big), & \text{CASE } 1 \\ 3\big(r_j(x) - r_{j-1}(x)\big), & \text{CASE } 2 \\ 3\big(r_j(x) - r_{j-1}(x)\big), & \text{CASE } 3 \end{cases}$$

$$\begin{aligned} \hat{c}_{\text{SPLAY}_i} &= \sum_{1 \leq j \leq l} \hat{c}_{\text{ITER}_j} \\ &= \sum_{1 \leq j \leq l} c_{\text{ITER}_j} + \left(\Phi_{\text{ITER}_j} - \Phi_{\text{ITER}_{j-1}}\right) \\ &\leq 3\big(r_{\text{ITER}_l}(x) - r_{\text{ITER}_0}(x)\big) \; {\color{red}+1} \\ &= 3\big(\log n - r_{\text{ITER}_0}(x)\big) + 1 \end{aligned}$$

$$\hat{c}_{\text{ITER}_j} \leq \begin{cases} 0, & \text{CASE } 0 \\ 1 + 3\big(r_j(x) - r_{j-1}(x)\big), & \text{CASE } 1 \\ 3\big(r_j(x) - r_{j-1}(x)\big), & \text{CASE } 2 \\ 3\big(r_j(x) - r_{j-1}(x)\big), & \text{CASE } 3 \end{cases}$$

$$\begin{aligned} \hat{c}_{\text{SPLAY}_i} &= \sum_{1 \leq j \leq l} \hat{c}_{\text{ITER}_j} \\ &= \sum_{1 \leq j \leq l} c_{\text{ITER}_j} + \left(\Phi_{\text{ITER}_j} - \Phi_{\text{ITER}_{j-1}}\right) \\ &\leq 3\big(r_{\text{ITER}_l}(x) - r_{\text{ITER}_0}(x)\big) + 1 \\ &= 3\big(\log n - r_{\text{ITER}_0}(x)\big) + 1 \\ &\leq 3 \log n + 1 \\ &= O(\log n) \end{aligned}$$

Theorem (Balance Theorem)

$$\sum_{1 \le i \le m} c_{\text{SPLAY}_i} = O\Big((m+n)\log n\Big)$$

Theorem (Balance Theorem)

$$\sum_{1 \le i \le m} c_{\text{SPLAY}_i} = O\Big((m+n)\log n\Big)$$

Proof.

$$\sum_{1 \le i \le m} c_{\text{SPLAY}_i} = \sum_{1 \le i \le m} \hat{c}_{\text{SPLAY}_i} + \Big(\Phi_{\text{SPLAY}_0} - \Phi_{\text{SPLAY}_m}\Big)$$

□

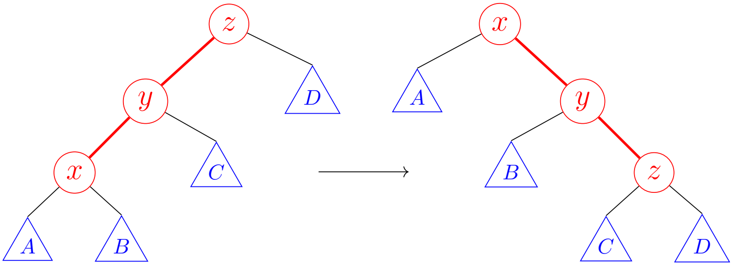
Theorem (Balance Theorem)

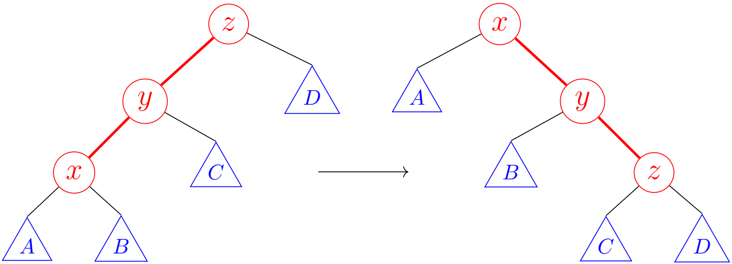
$$\sum_{1 \le i \le m} c_{\text{SPLAY}_i} = O\Big((m+n)\log n\Big)$$

Proof.

$$\begin{aligned} \sum_{1 \le i \le m} c_{\text{SPLAY}_i} &= \sum_{1 \le i \le m} \hat{c}_{\text{SPLAY}_i} + \Big(\Phi_{\text{SPLAY}_0} - \Phi_{\text{SPLAY}_m}\Big) \\ &\le m \log n + \ n \log n \\ &= (m+n)\log n \end{aligned}$$

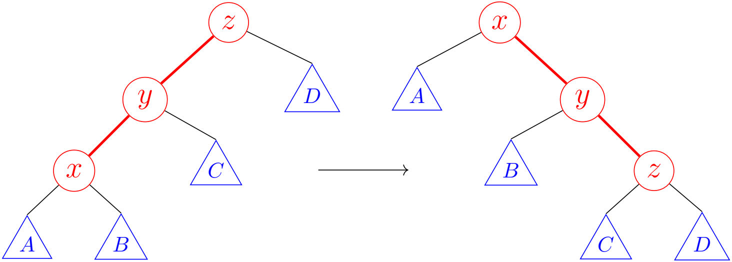
□


z
y
x
A
B
C
D
x
A
y
B
z
C
D



MTR (Move To Root) heuristic:

Keeping rotate the edge joining x to its parent.



MTR (Move To Root) heuristic:

Keeping rotate the edge joining x to its parent.

Does this work?

$$\Phi = \sum_{x \in T} r(x)$$

$$\Phi = \sum_{x \in T} r(x)$$

$\textsc{Splay}(x)$

$\textsc{Splay}(x)$

$\textsc{Search}(x, t)$

$\textsc{Insert}(x, t)$

$\textsc{Delete}(x, t)$

$\textsc{Join}(t_1, t_2)$

$\textsc{Split}(x, t)$

$\textsc{Splay}(x)$

$\textsc{Search}(x, t)$

$\textsc{Insert}(x, t)$

$\textsc{Delete}(x, t)$

$\textsc{Join}(t_1, t_2)$

$\textsc{Split}(x, t)$

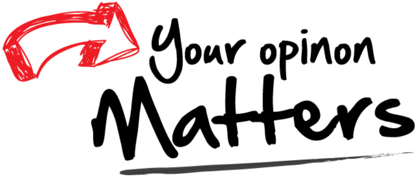
Self-Adjusting Binary Search Trees

DANIEL DOMINIC SLEATOR AND ROBERT ENDRE TARJAN

AT&T Bell Laboratories, Murray Hill, NJ

Abstract. The *splay* tree, a self-adjusting form of binary search tree, is developed and analyzed. The binary search tree is a data structure for representing tables and lists so that accessing, inserting, and deleting items is easy. On an *n*-node splay tree, all the standard search tree operations have an amortized time bound of $O(\log n)$ per operation, where by "amortized time" is meant the time per operation averaged over a worst-case sequence of operations. Thus splay trees are as efficient as balanced trees when total running time is the measure of interest. In addition, for sufficiently long access sequences, splay trees are as efficient, to within a constant factor, as static optimum search trees. The efficiency of splay trees comes not from an explicit structural constraint, as with balanced trees, but from applying a simple restructuring heuristic, called *splaying*, whenever the tree is accessed. Extensions of splaying give simplified forms of two other data structures: lexicographic or multidimensional search trees and link/cut trees.


THANK YOU



Office 302

Mailbox: H016

hfwei@nju.edu.cn