3-2 Amortized Analysis

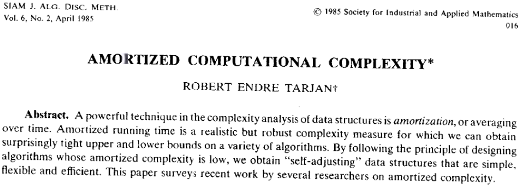
(Part II: Advanced Examples)

Hengfeng Wei

hfwei@nju.edu.cn

October 15, 2018

Robert Tarjan



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

© 1985 Society for Industrial and Applied Mathematics
016

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

What work are you proudest of?

What work are you proudest of?

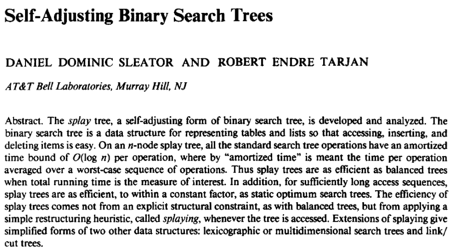
Proudest? It's hard to choose.

What work are you proudest of?

Proudest? It's hard to choose.

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

that Daniel 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

$\textsc{Splay}(x, T)$:

Moving node x to the ***root*** of the tree T by $\cdots$

$\textsc{Splay}(x, T)$:

Moving node x to the *root* of the tree T by $\cdots$

$\textsc{Search}(x, T)$ RETURN x^* / Λ

$\textsc{Insert}(x, T)$ ASSUME $x \notin T$

$\textsc{Delete}(x, T)$ ASSUME $x \in T$

$\textsc{Splay}(x, T)$:

Moving node x to the *root* of the tree T by $\cdots$

$$\begin{array}{rl} \textsc{Search}(x, T) & \textsf{RETURN } x^* / \Lambda \\ \textsc{Insert}(x, T) & \textsf{ASSUME } x \notin T \\ \textsc{Delete}(x, T) & \textsf{ASSUME } x \in T \end{array}$$

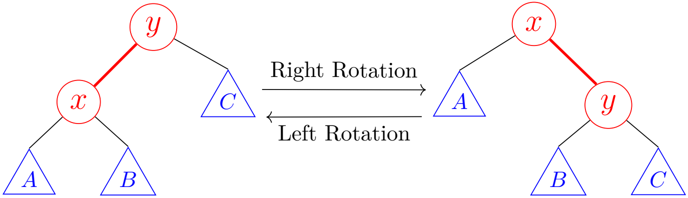
$$\begin{array}{rl} T \gets \textsc{Join}(T_1, T_2) & \textsf{ASSUME } x \in T_1 < y \in T_2 \\ (T_1, T_2) \gets \textsc{Split}(x, T) & \textsf{RETURN } x \in T_1 \le i \land y \in T_2 > i \end{array}$$

$\textsc{Splay}(x, T):$

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

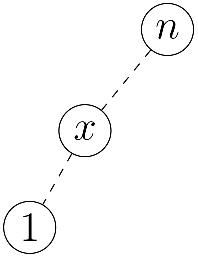
$\textsc{Splay}(x, T):$

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



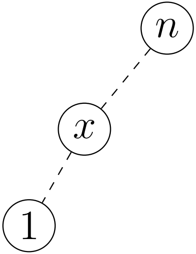
A chain of length n

A sequence of n SPLAY



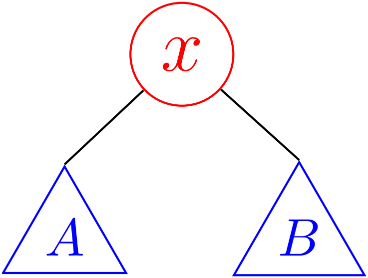
A chain of length n

A sequence of n Splay

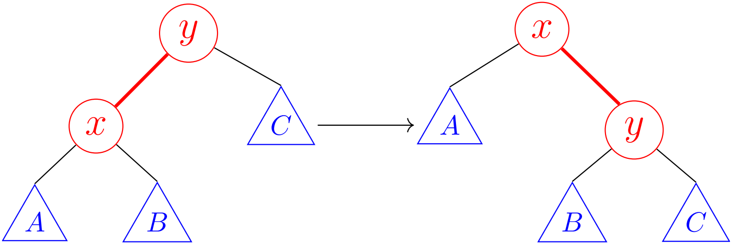


$$\sum_{i=1}^{n} c_i = \Theta(n^2)$$

$$\bar{c_i} = \Theta(n)$$

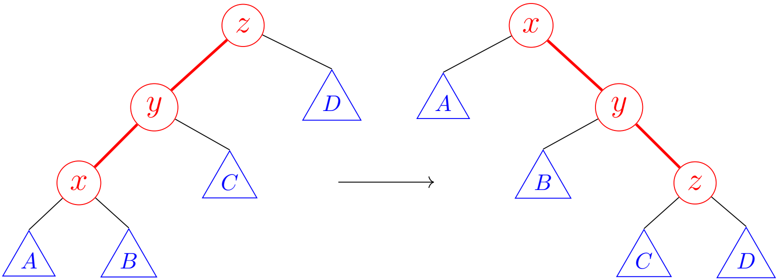


CASE 0: x is the root



CASE 1: zig (zag)

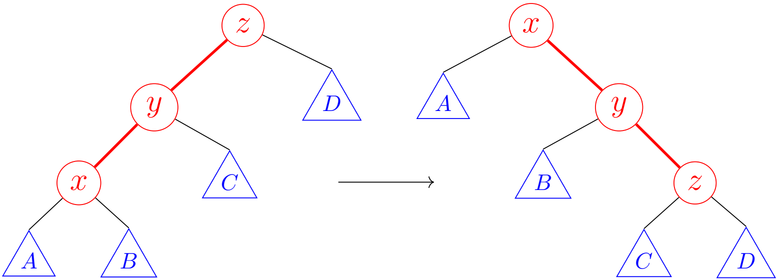
$y = p(x)$ is the root



CASE 2: zig-zig (zag-zag)

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

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

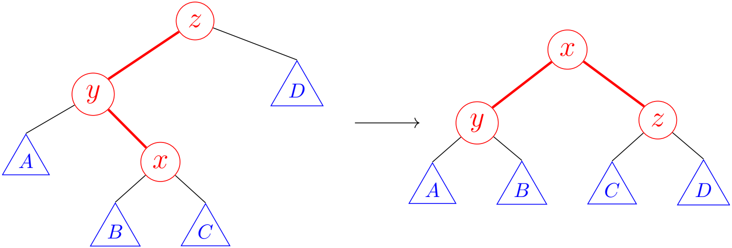


CASE 2: zig-zig (zag-zag)

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

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

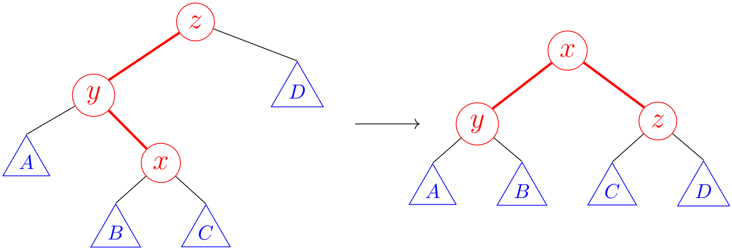
$$(1) : y - z \qquad (2) : x - y$$



CASE 3: zig-zag (zag-zig)

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

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



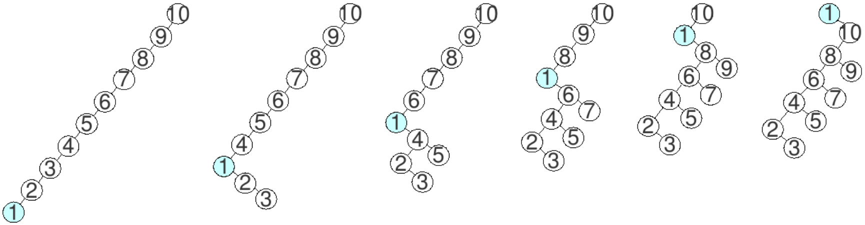
CASE 3: zig-zag (zag-zig)

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

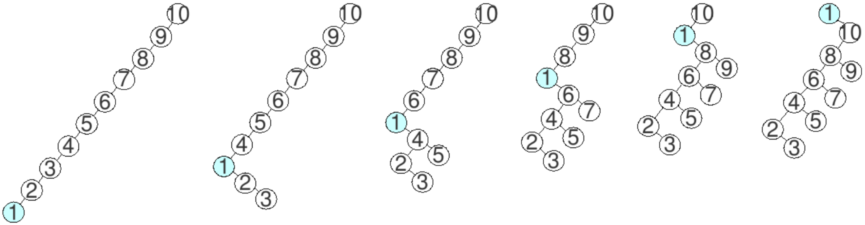
$$x = rc(y) \qquad y = lc(z)$$

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

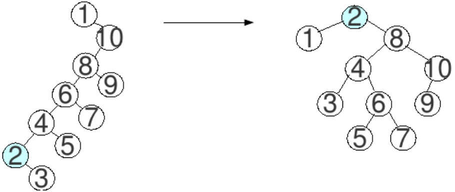
```
procedure SPLAY(x, T)
    while x ≠ T.root do                    ▷ Case 0
        switch ··· do
            case 1 : zig
                ···
                return
            case 2 : zig-zig
                ···
            case 3 : zig-zag
                ···
```



$\text{SPLAY}(1)$



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

Amortized Analysis of SPLAY

A splay tree T of n-node

An arbitrary sequence of m SPLAY

$\#$ of rotations

Amortized Analysis of Splay

A splay tree T of n-node

An arbitrary sequence of m Splay

$\#$ of rotations

Theorem

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

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

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

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

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

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

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

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

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

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

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

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

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

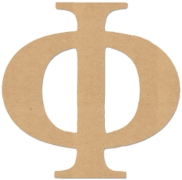
$$\underbrace{\Phi(D_0) - \Phi(D_n)}_{\text{net decrease in potential}} \le \square \implies \boxed{\sum_{1 \le i \le n} c_i \le \left(\sum_{1 \le i \le n} \hat{c}_i \right) + \square}$$

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

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

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

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



$s(x)$: # 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

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

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

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

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

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

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

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

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

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

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

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

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

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

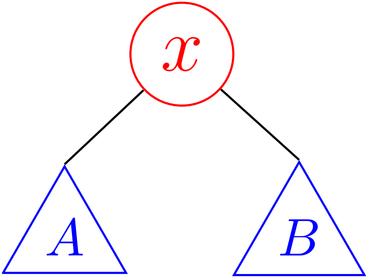
By Case Analysis.

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

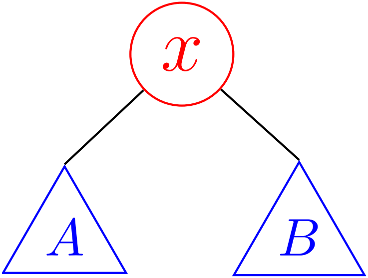
By Case Analysis.

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

Remember: ITER

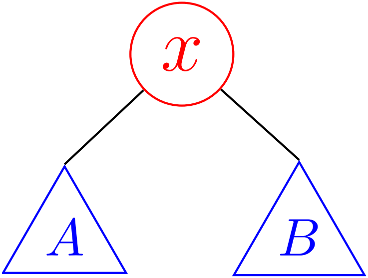


CASE 0



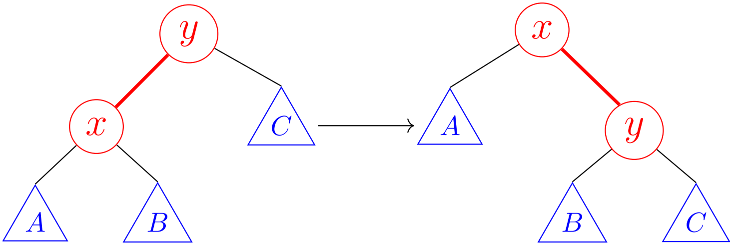
CASE 0

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



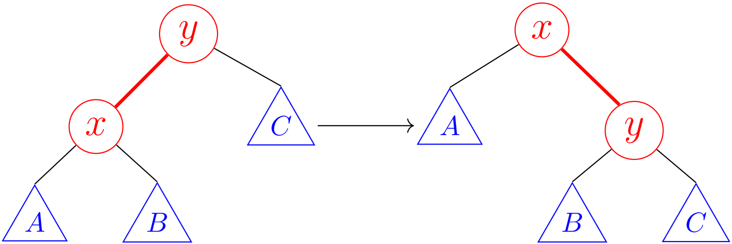
CASE 0

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



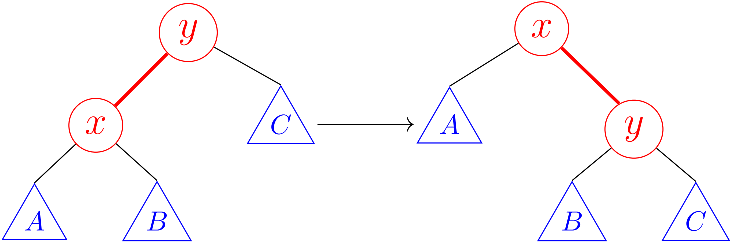
CASE 1: zig

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



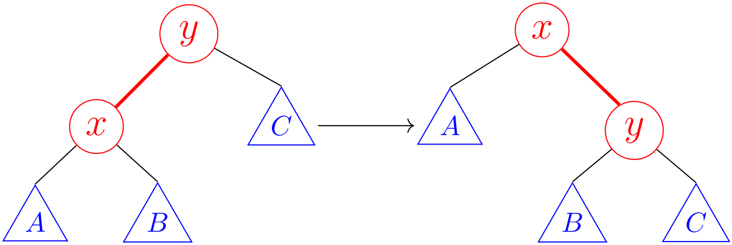
CASE 1: zig

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



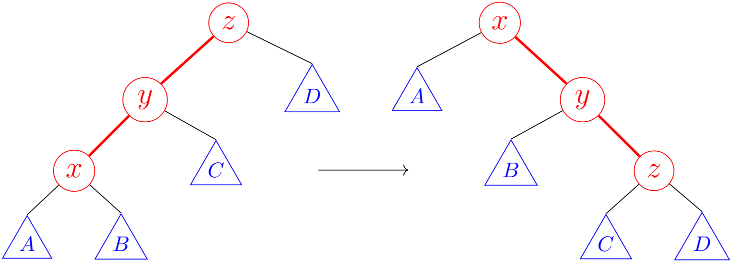
CASE 1: zig

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



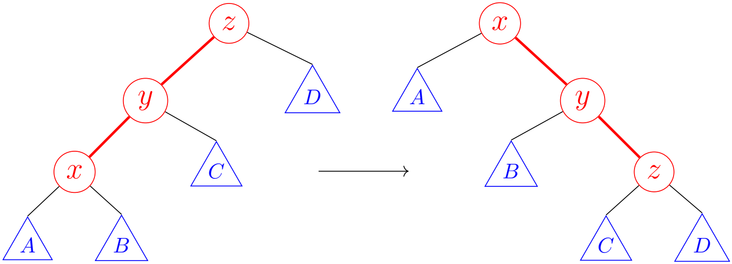
CASE 1: zig

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



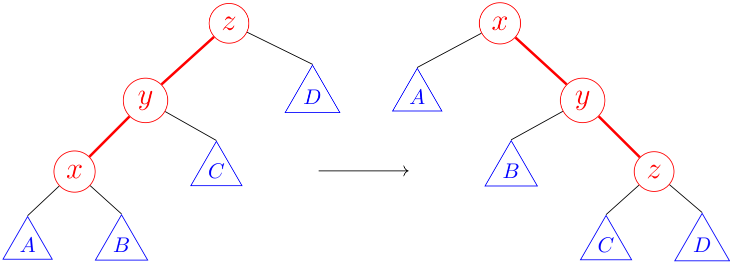
CASE 2: zig-zig

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



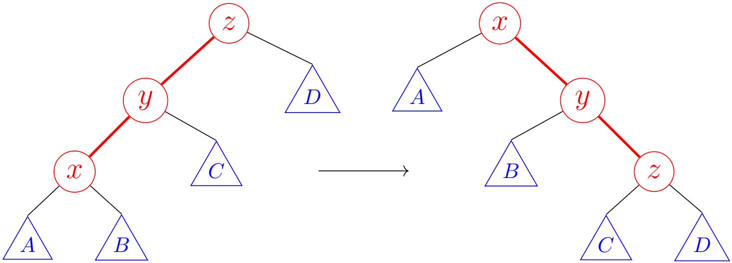
Case 2: zig-zig

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



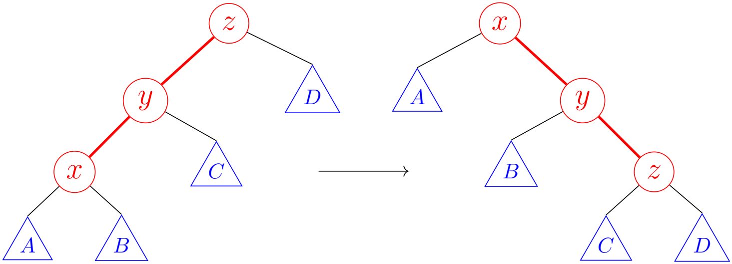
CASE 2: zig-zig

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



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

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

$$r_{j-1}(x) + r_j(z) = \log s_{j-1}(x) + \log s_i(z)$$

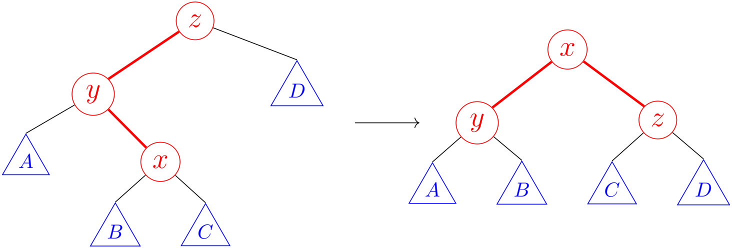
$$
\begin{aligned}
r_{j-1}(x) + r_j(z) &= \log s_{j-1}(x) + \log s_i(z) \\
&\leq 2 \log \left(\frac{s_{j-1}(x) + s_j(z)}{2} \right)
\end{aligned}
$$

$$
\begin{aligned}
r_{j-1}(x) + r_j(z) &= \log s_{j-1}(x) + \log s_i(z) \\
&\leq 2 \log \left(\frac{s_{j-1}(x) + s_j(z)}{2} \right) \\
&\leq 2 \log \left(\frac{s_j(x)}{2} \right)
\end{aligned}
$$

$$
\begin{aligned}
r_{j-1}(x) + r_j(z) &= \log s_{j-1}(x) + \log s_i(z) \\
&\leq 2 \log \left(\frac{s_{j-1}(x) + s_j(z)}{2} \right) \\
&\leq 2 \log \left(\frac{s_j(x)}{2} \right) \\
&= 2 \log s_j(x) - 2 \\
&= 2 r_j(x) - 2
\end{aligned}
$$

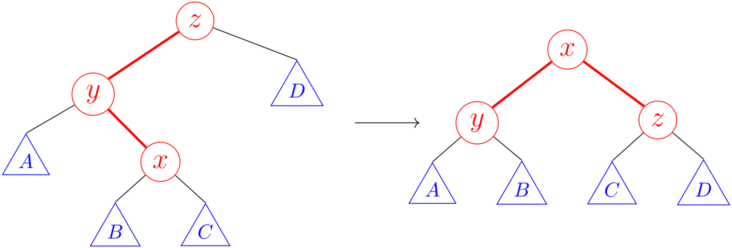
$$
\begin{aligned}
r_{j-1}(x) + r_j(z) &= \log s_{j-1}(x) + \log s_i(z) \\
&\leq 2 \log \left(\frac{s_{j-1}(x) + s_j(z)}{2} \right) \\
&\leq 2 \log \left(\frac{s_j(x)}{2} \right) \\
&= 2 \log s_j(x) - 2 \\
&= 2 r_j(x) - 2
\end{aligned}
$$

$$
r_j(z) \leq 2 r_j(x) - r_{j-1}(x) - 2
$$



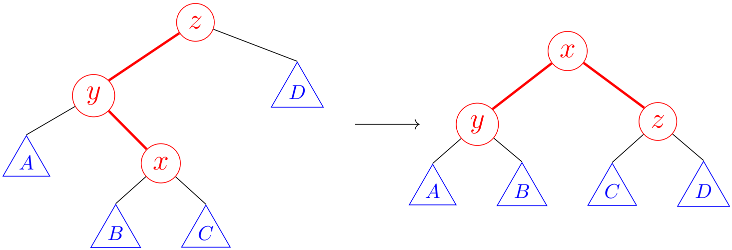
CASE 3: zig-zag

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



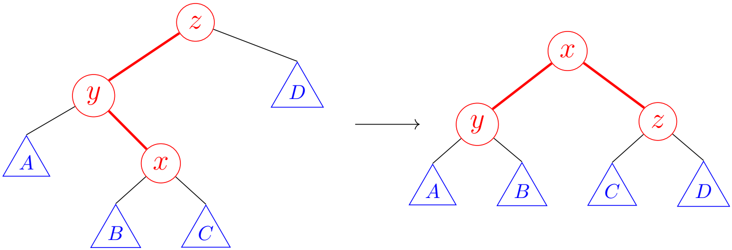
CASE 3: zig-zag

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



CASE 3: zig-zag

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



CASE 3: zig-zag

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

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

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

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

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

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

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

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

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

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

Theorem (Balance Theorem)

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

Proof.

$$\sum_{1 \le i \le m} c_{\text{SPLAY}_i} = \left(\sum_{1 \le i \le m} \hat{c}_{\text{SPLAY}_i} \right) + \big(\underbrace{\Phi_{\text{SPLAY}_0} - \Phi_{\text{SPLAY}_m}}_{\text{net decrease in potential}} \big)$$

□

Theorem (Balance Theorem)

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

Proof.

$$\sum_{1 \le i \le m} c_{\textsc{Splay}_i} = \left(\sum_{1 \le i \le m} \hat{c}_{\textsc{Splay}_i}\right) + \Big(\underbrace{\Phi_{\textsc{Splay}_0} - \Phi_{\textsc{Splay}_m}}_{\text{net decrease in potential}}\Big)$$

□

Theorem (Balance Theorem)

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

Proof.

$$\sum_{1 \le i \le m} c_{\text{SPLAY}_i} = \left(\sum_{1 \le i \le m} \hat{c}_{\text{SPLAY}_i} \right) + \Big(\underbrace{\Phi_{\text{SPLAY}_0} - \Phi_{\text{SPLAY}_m}}_{\text{net decrease in potential}} \Big)$$

$$\le m \log n + n \log n$$

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

□

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

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

$\textsc{Splay}(x)$

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

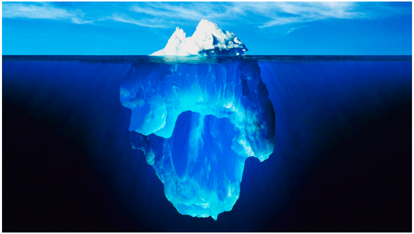
$\textsc{Insert}(x, t)$

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

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

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

$\textsc{Splay}(x)$



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

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

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

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

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

$\textsc{Splay}(x)$

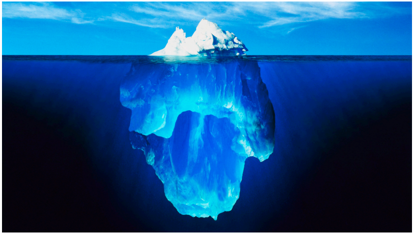
$\textsc{Search}(x, t)$

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

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

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

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



Self-Adjusting Binary Search Trees

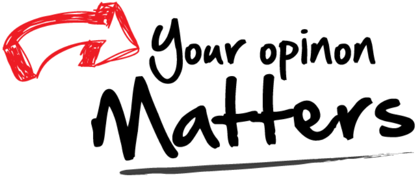
DANIEL DOMINIC SLEATOR AND ROBERT ENDRE TARJAN

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.

“Move-to-Front” (MTF) List


THANK YOU



Office 302

Mailbox: H016

hfwei@nju.edu.cn