Binary Search Trees: (Optional) Splay Tree Analysis

Daniel Kane

Department of Computer Science and Engineering
University of California, San Diego

Data Structures Fundamentals
Algorithms and Data Structures

Learning Objectives

- Prove the amortized runtime of a splay tree.
- Know other bounds on splay tree runtime.

Last Time

Analyzed splay trees given

Theorem

The amortized cost of doing $O(D)$ work and then splaying a node of depth D is $O(\log(n))$ where n is the total number of nodes.

Last Time

Analyzed splay trees given

Theorem

The amortized cost of doing $O(D)$ work and then splaying a node of depth D is $O(\log(n))$ where n is the total number of nodes.

Today we prove it.

Amortized Analysis

Need to amortize. Pick correct potential function.

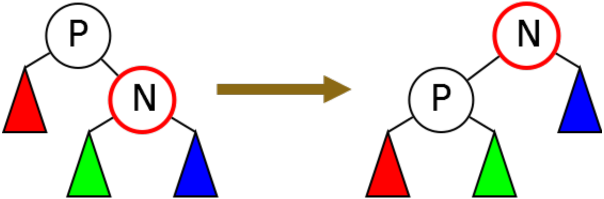
Rank

$R(N) = \log_2(\text{Size of subtree of } N)$.
Recall the size of the subtree of N is the total number of descendants of N.
Potential function

$$\Phi = \sum_N R(N).$$

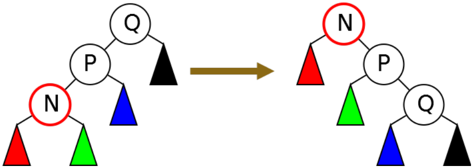
Zig Analysis

$$\begin{aligned}\Delta\Phi &= R'(N) + R'(P) - R(N) - R(P) \\ &= R'(P) - R(N) \\ &\leq R'(N) - R(N).\end{aligned}$$



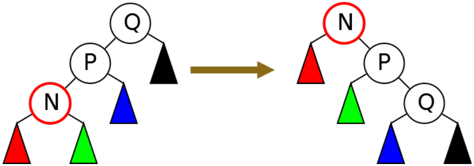
Zig-Zig Analysis

$$\begin{aligned}\Delta\Phi &= R'(N) + R'(P) + R'(Q) \\ &\quad - R(N) - R(P) - R(Q) \\ &= (R'(P) - R(P)) + (R'(Q) - R(N)) \\ &\leq 3(R'(N) - R(N)) - 2\end{aligned}$$



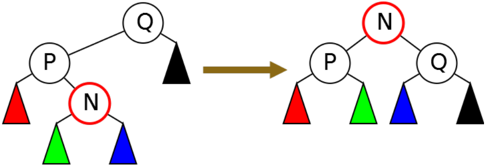
Why?

- $R(Q) = R'(N)$ bigger than any other term.
- $\text{Size}(N) + \text{Size}'(Q) = \text{Size}(Q) - 1$.
- So $R(N) + R'(Q) \leq 2R'(N) - 2$.



Zig-Zag Analysis

$$\begin{aligned}
\Delta\Phi &= R'(N) + R'(P) + R'(Q) \\
&\quad - R(N) - R(P) - R(Q) \\
&= (R'(P) - R(P)) + (R'(Q) - R(N)) \\
&\leq 2(R'(N) - R(N)) - 2
\end{aligned}$$



Total Change

$$
\begin{aligned}
\Delta\Phi &\leq 3(R_k(N) - R_{k-1}(N)) - 2 \\
&\qquad + 3(R_{k-1}(N) - R_{k-2}(N)) - 2 + \cdots \\
&= 3(R'(N) - R(N)) - \mathrm{Depth}(N) \\
&= O(\log(n)) - \mathrm{Work}
\end{aligned}
$$

Other Bounds

Splay trees have many other wonderful properties.

Weighted Nodes

If you assign weights so that

$$\sum_N \mathrm{wt}(N) = 1,$$

accessing N costs amortized $O(\log(1/\mathrm{wt}(N)))$ time.

Weighted Nodes

If you assign weights so that

$$\sum_N \mathrm{wt}(N) = 1,$$

accessing N costs amortized $O(\log(1/\mathrm{wt}(N)))$ time. So if you only access high weight nodes, it's much quicker.

Dynamic Finger

Amortized cost of accessing node $O(\log(D+1))$ where D is distance (in terms of the ordering) between last access and current access.

Working Set Bound

Amortized cost of accessing N is $O(\log(t+1))$ where t is time since N was last accessed.

Dynamic Optimality Conjecture

It is conjectured that for any sequence of binary search tree operations that a splay tree does at most a constant factor more work than the best dynamic search tree for that sequence.

Conclusion

Splay Trees

- Require $O(\log(n))$ amortized time per operation.
- Can be much better than this if queries have extra structure (call some nodes more frequently, calls nearby nodes, etc.).