

Notes for Lecture 23

In this lecture we analyze the splay tree algorithm presented in the last lecture. Splay tree algorithm is an elegant algorithm for maintaining a dictionary which stores its keys in a binary search tree. The algorithm attempts to keep the tree balanced and can be thought of as an analogue of the move-to-front algorithm from one of the previous lectures. The main intuition of the analysis is as follows: The “move-to-root” action of the algorithm results in approximately a doubling of the work required to perform the search part of the operation required to perform the FIND, INSERT, or DELETE action. When the search is short, this doubling does not add significantly to the overall amortized cost, but when the search is long, the resulting tree is better balanced than the tree we began with. We will define an appropriate *potential* function to measure the effect of each step on the tree balance.

In order to define our potential function, we need some preliminary definitions:

For each node X , we define $s(X)$ as the number of nodes in the subtree rooted at X , and the *rank*, $r(X)$ of the node X as $\lg(s(X))$, where the \lg stands for log to the base 2.

The *potential* of the tree T is $\Phi(T) = \sum_X r(X)$.

It is easy to see that the potential function Φ of an n -node tree T is minimized by a perfectly balanced tree for which $\Phi(T)$ is approximately n , and maximized by a tree in which every right child is a leaf, for which $\Phi(T) \approx n \lg(n)$.

We set the following costs for the operations involved in splay algorithm.

$$\begin{aligned} \text{ZIG} &\rightarrow 1 \\ \text{ZIG-ZIG} &\rightarrow 2 \\ \text{ZIG-ZAG} &\rightarrow 2 \end{aligned}$$

Note that the last two operations involved two rotations and the first one only one rotation. We define the *amortized cost* of an operation as its cost plus the change in potential.

We show that in an n -node tree the amortized cost of a splay operation is $O(\lg n)$. To show this we consider the amortized cost of a ZIG, a ZIG-ZIG, and ZIG-ZAG, and sum up to get an upper-bound on the overall amortized cost. We will use un-primed notation ($r(X), r(Y), s(X)$ etc.) to represent quantities before the operation and primed notation ($r'(X), r'(Y), s'(X)$ etc.) to represent the quantities after the operation. We will show that the amortized cost of ZIG is at most $1 + 2(r'(X) - r(X))$, and the amortized cost for ZIG-ZIG and ZIG-ZAG is at most $3(r'(X) - r(X))$.

ZIG:

Consider the ZIG operation in Fig. . Notice that the ranks of the nodes in the subtrees A, B , and C remain unchanged, and the ranks of only nodes X and Y change. Hence we have

$$\text{Amortized cost} = 1 + r'(X) + r'(Y) - r(X) - r(Y),$$

where the 1 is the cost of the ZIG operation itself. Since $r(Y) > r(X)$ and $r'(X) > r'(Y)$, this is upper bounded by $1 + 2(r'(X) - r(X))$.

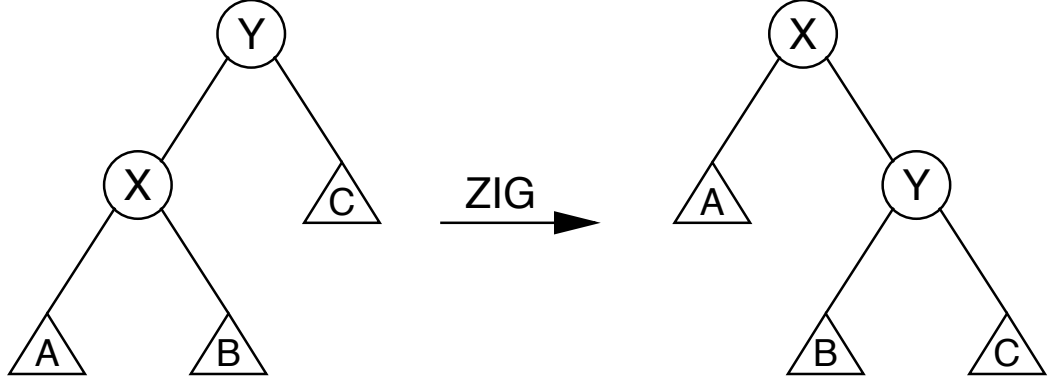


Figure 1: ZIG: In this and the following figures, circles represent nodes, and triangles represent (potentially empty) subtrees

ZIG-ZIG:

The ZIG-ZIG operation is shown in Fig. . The amortized cost of this is

$$\text{Amortized cost} = 2 + r'(X) + r'(Y) + r'(Z) - r(X) - r(Y) - r(Z).$$

We note that $r'(X) = r(Z)$, $r(Y) > r(X)$, and $r'(Y) < r'(X)$. This allows us to bound the above by

$$\text{Amortized cost} < 2 + r'(X) + r'(Z) - 2r(X).$$

In order to bound $r'(Z)$ in terms of $r(X)$ and $r'(X)$, we first note that

$$s(X) + s'(Z) \leq s'(X). \tag{1}$$

Using the concavity of \lg function, for any $a, b > 0$, we can write

$$\frac{\lg(a) + \lg(b)}{2} \leq \lg\left(\frac{a+b}{2}\right).$$

This implies that if $a + b \leq c$, then

$$\lg(a) + \lg(b) \leq 2\lg(c) - 2. \tag{2}$$

Now applying the above inequality to (1), we get

$$r(X) + r'(Z) \leq 2r'(X) - 2.$$

Thus it follows that

$$\text{Amortized cost} < 3(r'(X) - r(X)).$$

ZIG-ZAG:

Fig. shows the ZIG-ZAG operation. The amortized cost is given by

$$\text{Amortized cost} = 2 + r'(X) + r'(Y) + r'(Z) - r(X) - r(Y) - r(Z).$$

We note that $r'(X) = r(Z)$ and $r(Y) > r(X)$. Also, since $s'(Y) + s'(Z) \leq s'(X)$, using (2), we have

$$r'(Y) + r'(Z) \leq 2r'(X) - 2.$$

Hence

$$\text{Amortized cost} < 2(r'(X) - r(X)).$$

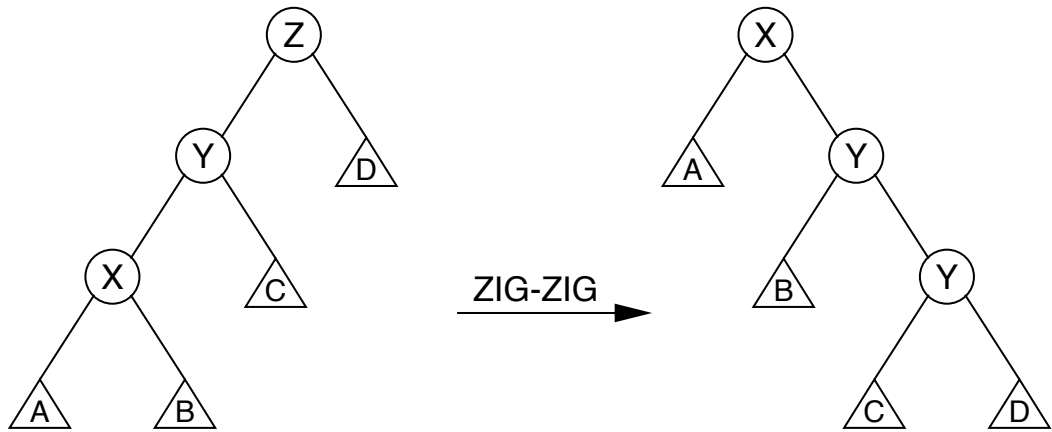


Figure 2: ZIG-ZIG

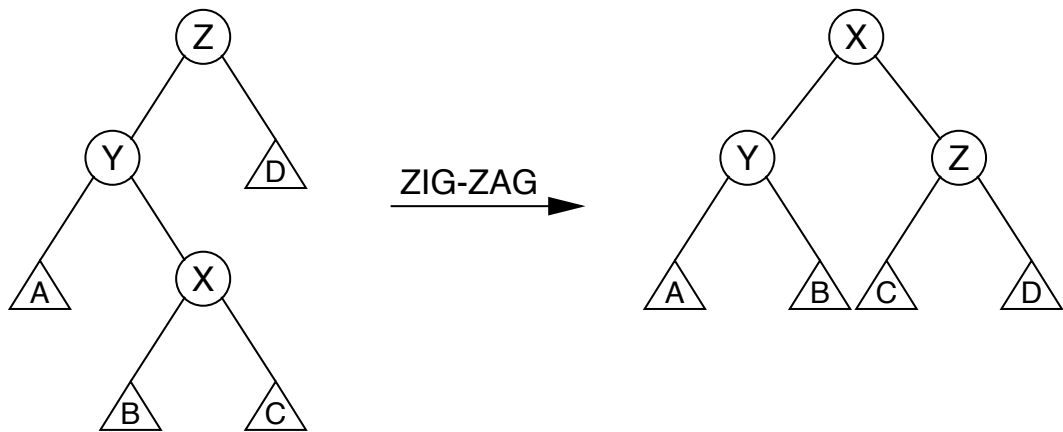


Figure 3: ZIG-ZAG

Therefore, the amortized cost of an entire splay operation is at most 1 + thrice the total increase in rank of X as it moves from its initial location to the root. Since the rank of the root is always $\lg n$, an upperbound on the amortized cost of an splay operation is $1 + 3 \lg n$.

Now we are ready to prove the main result.

Theorem 1. *The (actual) cost of a sequence of m splay operations on an n -node tree is $O(m + n) \lg n$ (where n is the maximum number of nodes).*

Proof. Let a_j be the amortized cost of the j -th splay operation, t_j , the actual cost, Φ_j the potential *after* the operation, and Φ_{j-1} the potential *before* the operation. Hence

$$\begin{aligned} \sum_{j=1}^m a_j &= \sum_{j=1}^m (t_j + \Phi_j - \Phi_{j-1}), \text{ so} \\ \sum_{j=1}^m t_j &= \sum_{j=1}^m (a_j + \Phi_{j-1} - \Phi_j) \\ &= \sum_{j=1}^m a_j + \Phi_0 - \Phi_m \\ &\leq \sum_{j=1}^m a_j + \Phi_0. \end{aligned}$$

But from the analysis preceding the statement of the theorem,

$$\sum_{j=1}^m a_j \leq m(1 + 3 \lg n).$$

Also, $\Phi_0 = O(n \lg n)$, since the maximum potential of an n -node tree is $O(n \lg n)$. Hence the total cost of a sequence of m splay operations is $O((m + n) \lg n)$. \square

The amortized cost of a FIND operation on an n -node tree does not exceed that of the corresponding splays. It can also be shown that the amortized cost of the corresponding splay is at most $O(\lg n)$. Thus the total cost of a sequence of m FIND, INSERT, and DELETE operations, where the i -th operation is on a tree of n_i nodes is $O(\sum_{i=1}^m \lg n_i) + O(n_0 \lg n_0)$.

Sleator and Tarjan[ST85] have explored other features of the splay tree algorithm. They prove the *static competitiveness theorem* which states that the splay tree algorithm performs within a constant factor of the off-line algorithm (which observes the sequence of operations before hand and constructs the optimal static binary search tree).

References

- [ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.