Splay Trees, I

This material is not covered in our textbook but you can read about splay trees in Section 7.3 of Data Structures and Their Algorithms by Lewis, Denenberg.

Splay trees are similar to red-black trees except that they guarantee good shape (small height) only on the average. They are simpler to program than red-black trees and have the additional advantage of giving faster access to items that are more frequently searched. The reason for both is that splay trees are self-adjusting.

Self-adjusting Binary Search Trees. Instead of explicitly maintaining the balance using additional information (such as the color of edges in the red-black tree), splay trees maintain balance implicitly through a self-adjusting mechanism. This means that good shape is a side-effect of the operations that are applied. These operations are applied while splaying a node, which means moving it up to the root of the tree. An example is shown in Figure 69. It turns out that single rotations do not imply good amortized performance. In other words, a single rotation does not necessarily improve the balance of the tree. We will see that that combinations of single rotations in pairs do imply good amortized performance. Aside from double rotations we use roller-coaster rotations that compose two single left or two single right rotations, as shown in Figure 72. The sequence of the two single rotations is important, namely first the higher then the lower one. Recall that Zig(κ) performs a single right rotation and returns the new root of the rotated subtree. The roller-coaster rotation to the right is then

\[ \text{Node } \ast \text{ZigZig(} \ast \text{k) \ return Zig(Zig(} \ast \text{k)).} \]

Function ZagZag is symmetric, exchanging left and right, and functions ZigZag and ZagZig are the two double rotations already used by red-black trees.

Splay. A splay operation finds an item and uses rotations to move the corresponding node up to the root position. Whenever possible a double rotation or a roller-coaster rotation is used.

Tree Splay(Tree \( \ast \) p, item \( \ast \) x)
\begin{verbatim}
if p = NULL then return NULL endif;
if x = p \rightarrow info then return p endif;
if x < p \rightarrow info then
    \mu = p \rightarrow \ell;
    if \mu = NULL then return \mu endif;
    if x < \mu \rightarrow info and \mu \rightarrow \ell \neq NULL then
        \mu \rightarrow \ell = Splay(\mu \rightarrow \ell, \mu);
        return ZigZig(\mu)
    else
        return Zig(\mu)
    endif;
else
    return ZigZag(\mu)
endif;
elseif x > \mu \rightarrow info and \mu \rightarrow r \neq NULL then
    \mu \rightarrow r = Splay(\mu \rightarrow r, \mu);
    return ZIGZAG(\mu)
else
    return Zig(\mu)
endif;
end.
\end{verbatim}

If \( \mu = NULL \) then function Splay takes no action at all and returns \( p \). The same is true if \( x \) is stored in \( p \). If \( x \) is stored in one of the children of \( p \) then it is moved to the root by a single rotation. Otherwise, it is splayed.
Amortized Cost. The amortized cost of an operation is the actual cost minus the cost for work put into improving the data structure. To quantify the cost for that work we introduce a measure of well-being for the data structure. We need definitions:

the size, $s(\nu)$, is the number of descendents of node $\nu$, including $\nu$;

the balance, $\beta(\nu)$, is twice the floor of the binary logarithm of the size, $\beta(\nu) = 2\lfloor \log_2 s(\nu) \rfloor$;

the potential, $\Phi$, of a tree or a collection of trees is the sum of balances over all nodes, $\Phi = \sum \beta(\nu)$;

the actual cost, $c_i$, of the $i$-th splay operation is 1 plus the number of single rotations (counting a double or roller-coaster rotation as two single rotations);

the amortized cost, $a_i$, of the $i$-th splay operation is $a_i = c_i + \Phi_i - \Phi_{i-1}$.

We have $\Phi_0 = 0$ for the empty tree and $\Phi_i \geq 0$ in general. This implies that the total actual cost does not exceed the total amortized cost, $\sum c_i = \sum a_i - \Phi_n + \Phi_0 \leq \sum a_i$. To get an intuitive feeling of what property of the splay tree the potential captures, we compute $\Phi$ for the two extreme cases shown in Figure 70. In the unbalanced case, we have $\Phi = 2 \sum_{i=1}^{n} \lfloor \log_2 i \rfloor$, which is $2n \log_2 n - O(n)$.

In the balanced case, we bound $\Phi$ from above by $2U(n)$, where $U(n) = U\left(\frac{\Phi}{2}\right) + \log_2 n$. We prove that $U(n) < 2n$ for the case when $n = 2^k$. Consider the perfectly balanced tree with $n$ leaves. The height of the tree is $k = \log_2 n$. We encode the term $\log_2 n$ of the recurrence relation by drawing the hook-like path from the root to the right child and then following left edges until we reach the leaf level. Each internal node encodes one of the recursively surfacing $\log$-terms by a hook-like path starting at that node. The paths are pairwise edge-disjoint, which implies that their total length is at most the number of edges in the tree, which is $2n - 2$.

Investment. The main part of the amortized time analysis is a detailed study of the three types of rotations: single, roller-coaster, and double rotation. We write $\beta(\nu)$ for the balance of a node $\nu$ before the rotation and $\beta'(\nu)$ for the balance after the rotation. Let $\nu$ be the lowest node involved in the rotation. The goal is to prove that the amortized cost of a roller-coaster and a double rotation is at most $3[\beta'(\nu) - \beta(\nu)]$ each, and that of a single rotation is at most $1 + 3[\beta'(\nu) - \beta(\nu)]$. Assuming this result we bound the amortized cost of a splay operation.

Investment Lemma. The amortized cost of splaying a node $\nu$ in a tree with root $\rho$ is at most $1 + 3[\beta(\rho) - \beta(\nu)]$.

Proof. A splay operation is a sequence of $k$ rotations that move $\nu$ from its initial to the root position. The first may be a single rotation, all others are either roller-coaster or double rotations. Let $\beta_i$ be the balance of $\nu$ after $i$ rotations. The amortized cost of the splay operation is

$$a \leq 1 + 3 \sum_{i=1}^{k} (\beta_i - \beta_{i-1})$$

$$= 1 + 3[\beta(\rho) - \beta(\nu)],$$

as claimed.

Before looking at the details of the three types of rotations, we prove that if two siblings have the same balance then their common parent has a larger balance. Because balances are even integers, this implies that the balance of the parent exceeds the balance of each child by at least 2.

Balance Lemma. If $\mu$ has children $\nu, \kappa$ and $\beta(\nu) = \beta(\kappa) = \beta$ then $\beta(\mu) \geq \beta + 2$.

Proof. By definition, $\beta(\nu) = 2\lfloor \log_2 s(\nu) \rfloor$ and therefore $s(\nu) \geq 2^{\beta/2}$. Similarly, $s(\kappa) \geq 2^{\beta/2}$. We have $s(\mu) = 1 + s(\nu) + s(\kappa) \geq 2^{1 + \beta/2}$, and therefore $\beta(\mu) \geq \beta + 2$. 

Figure 70: An extremely unbalanced tree to the left and a perfectly balanced tree to the right.
Single Rotation. See Figure 71 for an illustration of a single rotation. Its amortized cost is 1, for performing the rotation, plus the change in the potential:

\[ a = \begin{align*}
&= 1 + \beta'(v) + \beta'(\mu) - \beta(v) - \beta(\mu) \\
&= 1 + \beta'(\mu) - \beta(v) \\
&\leq 1 + 3[\beta'(v) - \beta(v)]
\end{align*} \]

because \( \beta'(v) = \beta(\mu), \beta'(\nu) \geq \beta'(\mu), \) and \( \beta'(\nu) - \beta(\nu) \geq 0. \)

![Figure 71: The size of \( \mu \) before the rotation is the same as the size of \( \nu \) after the rotation.](image)

Roller-Coaster Rotation. See Figure 72 for an illustration of a roller-coaster rotation. Its amortized cost is

\[ a = \begin{align*}
&= 2 + \beta'(v) + \beta'(\mu) + \beta'(\kappa) \\
&- \beta(v) - \beta(\mu) - \beta(\kappa) \\
&\leq 2 + 2[\beta'(v) - \beta(v)]
\end{align*} \]

because \( \beta'(v) = \beta(\kappa), \beta'(\nu) \geq \beta'(\mu), \beta'(\kappa), \) and \( \beta(\nu) \leq \beta(\mu). \) We distinguish two cases to prove that \( a \) is bounded from above by \( 3[\beta'(v) - \beta(v)] \). In both cases the drop in the potential pays for the two single rotations.

Case \( \beta'(\nu) > \beta(v) \). Then \( a \leq 3[\beta'(v) - \beta(v)], \) as before.

Case \( \beta'(\nu) = \beta(v) = \beta \). Then \( \beta(\mu) = \beta(\kappa) = \beta \). We have \( \beta'(\mu) < \beta'(v) \) or else \( \beta'(\kappa) < \beta'(v) \) by the Balance Lemma. Hence \( a \leq 0 = 3[\beta'(v) - \beta(v)] \).

![Figure 72: If in the middle tree the balance of \( \nu \) is the same as the balance of \( \mu \) then, by the Balance Lemma, the balance of \( \kappa \) is less than that common balance.](image)

Double Rotation. See Figure 73 for an illustration of a double rotation. Its amortized cost is

\[ a = \begin{align*}
&= 2 + \beta'(v) + \beta'(\mu) + \beta'(\kappa) \\
&- \beta(v) - \beta(\mu) - \beta(\kappa) \\
&\leq 2 + [\beta'(v) - \beta(v)]
\end{align*} \]

because \( \beta'(v) = \beta(\kappa), \beta'(\mu) \leq \beta(\mu), \) and \( \beta'(\nu) \geq \beta'(\kappa). \)

We again distinguish two cases to prove that \( a \) is bounded from above by \( 3[\beta'(v) - \beta(v)] \). In both cases the drop in the potential pays for the two single rotations.

Case \( \beta'(\nu) > \beta(v) \). Then \( a \leq 3[\beta'(v) - \beta(v)], \) as before.

Case \( \beta'(\nu) = \beta(v) = \beta \). Then \( \beta(\mu) = \beta(\kappa) = \beta \). We have \( \beta'(\mu) < \beta'(v) \) or else \( \beta'(\kappa) < \beta'(v) \) by the Balance Lemma. Hence \( a \leq 0 = 3[\beta'(v) - \beta(v)] \).

![Figure 73: In a double rotation the size of \( \mu \) decreases from before to after the operation.](image)