9 Splay Trees

Splay trees are similar to red-black trees except that they guarantee good shape (small height) only on the average. They are simpler to code 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. 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, as illustrated in Figure 33. A detailed analysis will

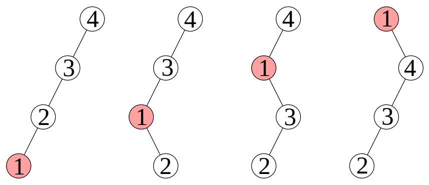

Figure 33: The node storing 1 is splayed using three single rotations.

reveal that single rotations do not imply good amortized performance but combinations of single rotations in pairs do. Aside from double rotations, we use *roller-coaster rotations* that compose two single left or two single right rotations, as shown in Figure 35. The sequence of the two single rotations is important, namely first the higher then the lower node. 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

```
Node $*$ ZIGZIG(Node $*\kappa$)
  return ZIG(ZIG($\kappa$)).
```

Function ZAGZAG is symmetric, exchanging left and right, and functions ZIGZAG and ZAGZIG are the two double rotations already used for 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. We dispense with special cases and show Function SPLAY for the case the search item x is less than the item in the root.

```
if $x < \varrho \to$ info then $\mu = \varrho \to \ell$;
  if $x < \mu \to$ info then
    $\mu \to \ell = $ SPLAY$(\mu \to \ell, x)$;
    return ZIGZIG$(\varrho)$
  elseif $x > \mu \to$ info then
    $\mu \to r = $ SPLAY$(\mu \to r, x)$;
    return ZIGZAG$(\varrho)$
  else
    return ZIG$(\varrho)$
endif.
```

If x is stored in one of the children of ϱ then it is moved to the root by a single rotation. Otherwise, it is splayed recursively to the third level and moved to the root either by a double or a roller-coaster rotation. The number of rotation depends on the length of the path from ϱ to x. Specifically, if the path is i edges long then x is splayed in $\lfloor i/2 \rfloor$ double and roller-coaster rotations and zero or one single rotation. In the worst case, a single splay operation takes almost as many rotations as there are nodes in the tree. We will see shortly that the amortized number of rotations is at most logarithmic in the number of nodes.

Amortized cost. Recall that the amortized cost of an operation is the actual cost minus the cost for work put into improving the data structure. To analyze the cost, we use a potential function that measures the well-being of the data structure. We need definitions:

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

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* Φ 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 a feeling for the potential, we compute Φ for the two extreme cases. Note first that the integral of the natural logarithm is $\int \ln x = x \ln x - x$ and therefore $\int \log_2 x = x \log_2 x - x/\ln 2$. In the extreme unbalanced case, the balance of the i-th node from the bottom is $2\lfloor \log_2 i \rfloor$ and the potential is

$$\Phi = 2 \sum_{i=1}^{n} \lfloor \log_2 i \rfloor = 2n \log_2 n - \mathrm{O}(n).$$

In the balanced case, we bound Φ from above by $2U(n)$, where $U(n) = 2U(\frac{n}{2}) + \log_2 n$. 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. We write $\beta(\nu)$ for the balance of a node ν before the rotation and $\beta'(\nu)$ for the balance after the rotation. Let ν 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 $1 + 3[\beta'(\nu) - \beta(\nu)]$. Summing these terms over the rotations of a splay operation gives a telescoping series in which all terms cancel except the first and the last. To this we add 1 for the at most one single rotation and another 1 for the constant cost in definition of actual cost.

INVESTMENT LEMMA. The amortized cost of splaying a node ν in a tree ϱ is at most $2 + 3[\beta(\varrho) - \beta(\nu)]$.

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 means that the balance of the parent exceeds the balance of its children by at least 2.

BALANCE LEMMA. If μ has children ν, κ 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 $s(\nu) \geq 2^{\beta/2}$. We have $s(\mu) = 1 + s(\nu) + s(\kappa) \geq 2^{1+\beta/2}$, and therefore $\beta(\mu) \geq \beta + 2$. ◻

Single rotation. The amortized cost of a single rotation shown in Figure 34 is 1 for performing the rotation plus the change in the potential:

$$\begin{aligned} a &= 1 + \beta'(\nu) + \beta'(\mu) - \beta(\nu) - \beta(\mu) \\ &\leq 1 + 3[\beta'(\nu) - \beta(\nu)] \end{aligned}$$

because $\beta'(\mu) \leq \beta(\mu)$ and $\beta(\nu) \leq \beta'(\nu)$.

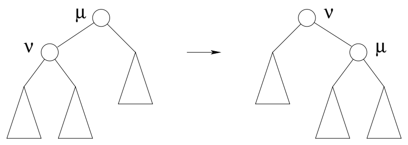

Figure 34: The size of μ decreases and that of ν increases from before to after the rotation.

Roller-coaster rotation. The amortized cost of a roller-coaster rotation shown in Figure 35 is

$$\begin{aligned} a &= 2 + \beta'(\nu) + \beta'(\mu) + \beta'(\kappa) \\ &\quad - \beta(\nu) - \beta(\mu) - \beta(\kappa) \\ &\leq 2 + 2[\beta'(\nu) - \beta(\nu)] \end{aligned}$$

because $\beta'(\kappa) \leq \beta(\kappa)$, $\beta'(\mu) \leq \beta'(\nu)$, and $\beta(\nu) \leq \beta(\mu)$. We distinguish two cases to prove that a is bounded from above by $3[\beta'(\nu) - \beta(\nu)]$. In both cases, the drop in the

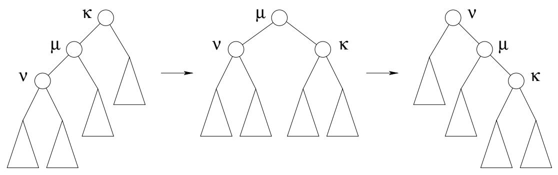

Figure 35: If in the middle tree the balance of ν is the same as the balance of μ then by the Balance Lemma the balance of κ is less than that common balance.

potential pays for the two single rotations.

`Case` $\beta'(\nu) > \beta(\nu)$. The difference between the balance of ν before and after the roller-coaster rotation is at least 2. Hence $a \leq 3[\beta'(\nu) - \beta(\nu)]$.

`Case` $\beta'(\nu) = \beta(\nu) = \beta$. Then the balances of nodes ν and μ in the middle tree in Figure 35 are also equal to β. The Balance Lemma thus implies that the balance of κ in that middle tree is at most $\beta - 2$. But since the balance of κ after the roller-coaster rotation is the same as in the middle tree, we have $\beta'(\kappa) < \beta$. Hence $a \leq 0 = 3[\beta'(\nu) - \beta(\nu)]$.

Double rotation. The amortized cost of a double rotation shown in Figure 36 is

$$\begin{aligned} a &= 2 + \beta'(\nu) + \beta'(\mu) + \beta'(\kappa) \\ &\quad - \beta(\nu) - \beta(\mu) - \beta(\kappa) \\ &\leq 2 + [\beta'(\nu) - \beta(\nu)] \end{aligned}$$

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

`Case` $\beta'(\nu) > \beta(\nu)$. The difference is at least 2, which implies $a \leq 3[\beta'(\nu) - \beta(\nu)]$, as before.

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

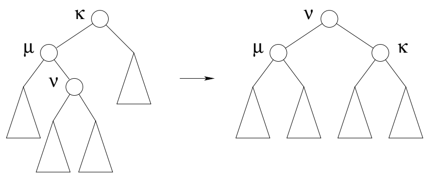

Figure 36: In a double rotation, the sizes of μ and κ decrease from before to after the operation.

Dictionary operations. In summary, we showed that the amortized cost of splaying a node ν in a binary search tree with root ϱ is at most $1 + 3[\beta(\varrho) - \beta(\nu)]$. We now use this result to show that splay trees have good amortized performance for all standard dictionary operations and more.

To **access** an item we first splay it to the root and return the root even if it does not contain x. The amortized cost is O$(\beta(\varrho))$.

Given an item x, we can **split** a splay tree into two, one containing all items smaller than or equal to x and the other all items larger than x, as illustrated in Figure 37. The amortized cost is the amortized cost for splaying plus the increase in the potential, which we denote as $\Phi' - \Phi$. Recall that the potential of a collection of trees is the sum of the balances of all nodes. Splitting the tree decreases the number of descendents and therefore the balance of the root, which implies that $\Phi' - \Phi < 0$. It follows that the amortized cost of a split operation is less than that of a splay operation and therefore in O$(\beta(\varrho))$.

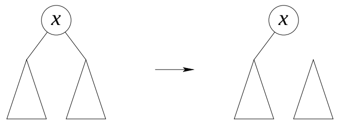

Figure 37: After splaying x to the root, we split the tree by unlinking the right subtree.

Two splay trees can be **joined** into one if all items in one tree are smaller than all items in the other tree, as illustrated in Figure 38. The cost for splaying the maximum

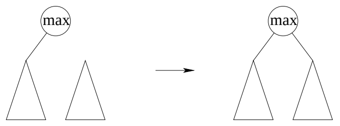

Figure 38: We first splay the maximum in the tree with the smaller items and then link the two trees.

in the first tree is O$(\beta(\varrho_1))$. The potential increase caused by linking the two trees is

$$\begin{aligned} \Phi' - \Phi &\leq 2\lfloor \log_2(s(\varrho_1) + s(\varrho_2)) \rfloor \\ &\leq 2\log_2 s(\varrho_1) + 2\log_2 s(\varrho_2). \end{aligned}$$

The amortized cost of joining is thus O$(\beta(\varrho_1) + \beta(\varrho_2))$.

To **insert** a new item, x, we split the tree. If x is already in the tree, we undo the split operation by linking the two trees. Otherwise, we make the two trees the left and right subtrees of a new node storing x. The amortized cost for splaying is O$(\beta(\varrho))$. The potential increase caused by linking is

$$\begin{aligned} \Phi' - \Phi &\leq 2\lfloor \log_2(s(\varrho_1) + s(\varrho_2) + 1) \rfloor \\ &= \beta(\varrho). \end{aligned}$$

The amortized cost of an insertion is thus O$(\beta(\varrho))$.

To **delete** an item, we splay it to the root, remove the root, and join the two subtrees. Removing x decreases the potential, and the amortized cost of joining the two subtrees is at most O$(\beta(\varrho))$. This implies that the amortized cost of a deletion is at most O$(\beta(\varrho))$.

Weighted search. A nice property of splay trees not shared by most other balanced trees is that they automatically adapt to biased search probabilities. It is plausible that this would be the case because items that are often accessed tend to live at or near the root of the tree. The analysis is somewhat involved and we only state the result. Each item or node has a positive weight, $w(\nu) > 0$, and we define $W = \sum_\nu w(\nu)$. We have the following generalization of the Investment Lemma, which we state without proof.

WEIGHTED INVESTMENT LEMMA. The amortized cost of splaying a node ν in a tree with total weight W is at most $2 + 3\log_2(W/w(\nu))$.

It can be shown that this result is asymptotically best possible. In other words, the amortized search time in a splay tree is at most a constant times the optimum, which is what we achieve with an optimum weighted binary search tree. In contrast to splay trees, optimum trees are expensive to construct and they require explicit knowledge of the weights.