splay trees

advanced data structures
Scapegoat (Lazy Height-Balanced) Trees

Finally, to handle both insertions and deletions efficiently, scapegoat trees use both of the previous techniques. We use partial rebuilding to re-balance the tree after insertions, and global rebuilding to re-balance the tree after deletions. Each search takes $O(\log n)$ time in the worst case, and the amortized time for any insertion or deletion is also $O(\log n)$. There are a few small technical details left (which I won't describe), but no new ideas are required.

Once we've done the analysis, we can actually simplify the data structure. It's not hard to prove that at most one subtree (the scapegoat) is rebuilt during any insertion. Less obviously, we can even get the same amortized time bounds (except for a small constant factor) if we only maintain the three integers in addition to the actual tree: the size of the entire tree, the height of the entire tree, and the number of marked nodes. Whenever an insertion causes the tree to become unbalanced, we can compute the sizes of all the subtrees on the search path, starting at the new leaf and stopping at the scapegoat, in time proportional to the size of the scapegoat subtree. Since we need that much time to re-balance the scapegoat subtree, this computation increases the running time by only a small constant factor! Thus, unlike almost every other kind of balanced trees, scapegoat trees require only $O(1)$ extra space.

**Rotations, Double Rotations, and Splaying**

Another method for maintaining balance in binary search trees is by adjusting the shape of the tree locally, using an operation called a rotation. A rotation at a node $x$ decreases its depth by one and increases its parent's depth by one. Rotations can be performed in constant time, since they only involve simple pointer manipulation.

![Figure 1](image.png)

**Figure 1.** A right rotation at $x$ and a left rotation at $y$ are inverses.

A splay tree is a binary search tree that is kept more or less balanced by splaying. Intuitively, after we access any node, we move it to the root with a splay operation. In more detail:
zig-zag rotation

Figure 2. A zig-zag at $x$. The symmetric case is not shown.
roller-coaster rotation

Figure 3. A right roller-coaster at $x$ and a left roller-coaster at $z$. 
splay operation

• whenever a node $x$ is accessed, it is splayed
• splaying means to rotate a node $x$ to the root, using a series of zig-zig and roller coaster rotations
• ... and possibly one single rotation when $x$ is the child of the root
measures

• $s(x)$ is size of a node $x$, the number of nodes in the subtree rooted at $x$
• $r(x)$ is the rank of $x$, the log base 2 of the size of $x$: $r(x) = \log_2 s(x)$
• the potential $\Phi$ of a tree is the sum of $r(x)$ for all nodes $x$ in the tree
• $\Phi(T) = \sum_{\text{nodes } x} r(x) = \sum_{x \in T} \log_2 s(x)$
main theorem

The Generalized Access Lemma. For any assignment of non-negative weights to the nodes, the amortized cost of a single rotation at any node \( x \) is at most \( 1 + 3r'(x) - 3r(x) \), and the amortized cost of a double rotation at any node \( v \) is at most \( 3r'(x) - 3r(x) \).

- this is used to show that the amortized cost of a splay operation is \( O(\lg n) \)
- the text refers to this as terms cancelling in a “telescoping sum”
- to illustrate that:
  - suppose that a splay of node \( x \) uses \( k \) rotations
  - rotations \( 1, \ldots, k-1 \) are double, and rotation \( k \) is single
  - let \( r^i(x) \) be the rank of \( x \) after the \( i \)th rotation
  - now add up the amortized costs given by the Generalized Access Lemma
telescoping sums

• $3(r^1(x) - r^0(x))$
• $\ldots + 3(r^2(x) - r^1(x))$
• $\ldots + 3(r^3(x) - r^2(x))$
• $\ldots$
• $\ldots + 3 \left( r^{k-1}(x) - r^{k-2}(x) \right)$
• $\ldots + 1 + 3 \left( r^k(x) - r^{k-1}(x) \right)$
• $= 1 + 3 \left( r^k(x) - r^0(x) \right)$
• $\leq 1 + 3r^k(x) \leq 1 + 3 \log_2 n$
proof part 1: single rotation

\[1 + \Phi' - \Phi = 1 + r'(x) + r'(y) - r(x) - r(y)\]  
\[\leq 1 + r'(x) - r(x)\]  
\[\leq 1 + 3r'(x) - 3r(x)\]  

[only \(x\) and \(y\) change rank]  
[\(r'(y) \leq r(y)\)]  
[\(r'(x) \geq r(x)\)]
proof part 2: zig-zag rotation

\[ 2 + \Phi' - \Phi \]

\[ = 2 + r'(w) + r'(x) + r'(z) - (w) - (x) - (z) \]

[only w, x, z change rank]

\[ \leq 2 + r'(w) + r'(x) + r'(z) - 2r(x) \]

[r(x) \leq (w) and r'(x) = (z)]

\[ = 2 + (r'(w) - r'(x)) + (r'(z) - r'(x)) + 2(r'(x) - r(x)) \]

\[ = 2 \left( l_g \frac{s'(w)}{s'(x)} + l_g \frac{s'(z)}{s'(x)} + 2(r'(x) - r(x)) \right) \]

\[ \leq 2 + 2l_g \frac{s'(x)/2}{s'(x)} + 2(r'(x) - r(x)) \]

[s'(w) + s'(z) \leq s'(x), lg is concave]

\[ = 2(r'(x) - r(x)) \]

[r'(x) \geq r(x)]

\[ \leq 3(r'(x) - r(x)) \]
proof part 3: roller coaster rotation

\[2 + \Phi' - \Phi\]
\[= 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)\]
\[\leq 2 + r'(x) + r'(z) - 2r(x)\]
\[= 2 + (r(x) - r'(x)) + (r'(z) - r'(x)) + 3(r'(x) - r(x))\]
\[= 2 + \lg \frac{s(x)}{s'(x)} + \lg \frac{s'(z)}{s'(x)} + 3(r'(x) - r(x))\]
\[\leq 2 + 2\lg \frac{s'(x)/2}{s'(x)} + 3(r'(x) - r(x))\]
\[= 3(r'(x) - r(x))\]
fact about $\log$ 

$$\frac{\log a + \log b}{2} \leq \log \frac{a + b}{2}$$

This is true because $\log$ (all logarithms are base 2) is concave – see next slide for idea
\[ \lg \left( \frac{a+b}{2} \right) \]

\[ \frac{\lg a + \lg b}{2} = \frac{\lg (a+b)}{2} \]

\[ \lg a \]

\[ \lg b \]

\[ \lg \text{function} \]

\[ a \]

\[ b \]
if \( a + b \leq c \) then \( \lg \frac{a}{c} + \lg \frac{b}{c} \leq 2 \lg \frac{c}{2} \)

• \( \lg \frac{a}{c} + \lg \frac{b}{c} = \lg a + \lg b - 2 \lg c \)

• \( \ldots = 2 \left[ \lg a + \lg b \right] - 2 \lg c \)

• \( \ldots \leq 2 \lg \frac{a + b}{2} - 2 \lg c \)

• \( \ldots \leq 2 \lg \frac{c}{2} - 2 \lg c \)

• \( \ldots = 2 \lg \frac{c}{2} \)

d by the fact about \( \lg \) shown earlier

since \( a + b \leq c \)