Lecture 16: Splay Trees

Handout

March 26, 2002

1 Splay trees

- Last time we started discussing splay trees:
  - Search trees that “magically” balance themselves (no rebalance information is stored) and have amortized $O(\log n)$ performance.
  - This is in contrast to red-black trees, which have $O(\log n)$ worst-case performance but are quite complicated and use extra balancing information (color of each node).

- Splay trees are normal binary search trees:
  - Binary tree with elements in nodes
  - If node $v$ holds element $e$ then
    * all elements in left subtree < $e$
    * all elements in left subtree > $e$

- All operations on Splay trees implemented using one basic operation, SPLAY:

  $\text{SPLAY}(x, T)$ searches for $x$ in $T$ and reorganizes tree such that $x$ (or min element > $x$ or max element < $x$) is in root

- Last time we saw how SEARCH, INSERT, and DELETE can be implemented using $O(1)$ splay operations and a few pointer manipulation.

- Implementation of SPLAY:
  - Search for $x$ like in normal search tree
  - Repeatedly rotate $x$ up until it becomes the root.
    We distinguish between three cases:
    1. $x$ is child of root (no grandparent): rotate($x$)

    e.g.

    \begin{center}
    \begin{tabular}{c}
    \begin{tikzpicture}
    \node (x) at (0,0) {x};
    \node (y) at (0.5,0.5) {y};
    \node (t1) at (-1,-1) {T1};
    \node (t2) at (0,-1) {T2};
    \node (t3) at (0.5,-1) {T3};
    \draw (x) -- (t1);
    \draw (x) -- (t2);
    \draw (x) -- (t3);
    \draw (y) -- (t2);
    \draw (y) -- (t3);
    \end{tikzpicture}
    \end{tabular}
    \hspace{1cm}
    \begin{tikzpicture}
    \node (x) at (0,0) {x};
    \node (y) at (0.5,0.5) {y};
    \node (t1) at (-1,-1) {T1};
    \node (t2) at (0,-1) {T2};
    \node (t3) at (0.5,-1) {T3};
    \draw (x) -- (t1);
    \draw (x) -- (t2);
    \draw (x) -- (t3);
    \end{tikzpicture}
    \end{center}
2. \(x\) has parent \(y\) and grandparent \(z\) and both \(x\) and \(y\) left (right) children: \text{rotate}(y) \text{ followed by rotate}(x)

\text{e.g.}

\begin{align*}
\begin{array}{c}
\text{T1} \\
\text{T2} \\
\text{T3} \\
\text{T4}
\end{array} \\
\begin{array}{c}
\text{x} \\
\text{y} \\
\text{z}
\end{array}
\quad \text{→} \quad 
\begin{array}{c}
\text{T1} \\
\text{T2} \\
\text{T3} \\
\text{T4}
\end{array} \\
\begin{array}{c}
\text{x} \\
\text{y} \\
\text{z}
\end{array}
\quad \text{→} \quad 
\begin{array}{c}
\text{T1} \\
\text{T2} \\
\text{T3} \\
\text{T4}
\end{array} \\
\begin{array}{c}
\text{x} \\
\text{y} \\
\text{z}
\end{array}
\end{align*}

3. \(x\) has parent \(y\) and grandparent \(z\) and one of \(x\) and \(y\) is a left child and the other is a right child: \text{rotate}(x) \text{ followed by rotate}(x)

\text{e.g.}

\begin{align*}
\begin{array}{c}
\text{T1} \\
\text{T2} \\
\text{T3} \\
\text{T4}
\end{array} \\
\begin{array}{c}
\text{x} \\
\text{y} \\
\text{z}
\end{array}
\quad \text{→} \quad 
\begin{array}{c}
\text{T1} \\
\text{T2} \\
\text{T3} \\
\text{T4}
\end{array} \\
\begin{array}{c}
\text{x} \\
\text{y} \\
\text{z}
\end{array}
\quad \text{→} \quad 
\begin{array}{c}
\text{T1} \\
\text{T2} \\
\text{T3} \\
\text{T4}
\end{array} \\
\begin{array}{c}
\text{x} \\
\text{y} \\
\text{z}
\end{array}
\end{align*}

- Note:
  - A SPLAY can take \(O(n)\) worst-case time (very unbalanced tree)
  - But as we saw last time, Splay trees somehow seem to stay nicely balanced

- Analysis:
  - We will use \text{accounting method} to show that all operations (SPLAY) take \(O(\log n)\) amortized time.
    - We will imagine that each node in tree has credits on it
    - We will use some credits to pay for (part of) rotations during a splay
    - Note that we will ignore cost of searching for \(x\), since the rotations cost at least as much as the search (⇒ if we can bound amortized rotation cost we also bound search cost).
    - Let \(T(x)\) be tree rooted at \(x\). We will maintain the \text{credit invariant} that each node \(x\) holds \(\mu(x) = \lfloor \log |T(x)| \rfloor\) credits.
    - We will prove the following lemma:

| Less than or equal to \(3(\mu(T) - \mu(x) + O(1))\) credits are needed to perform SPLAY \((x, T)\) operation and maintain credit invariant |

- Using this lemma we get that a SPLAY operation uses at most \(3\log n + O(1) = O(\log n)\) credits (time).
- Last time we discussed that apart from SPLAY cost, an \text{INSERT} or a \text{DELETE} requires us to insert at most \(O(\log n)\) extra credits (on the root) ⇒ \(O(\log n)\) amortized bound.
• Proof of lemma:
  
  – Let $\mu$ and $\mu'$ be the value of $\mu$ before and after a rotate operation in case 1, 2, or 3.
  – During a SPLAY operation we perform a number of, say $k \geq 0$, case 2 and 3 operations and possibly a case 1 operation.
  – We will show that the cost of one operation is:
    * Case 1: $3(\mu'(x) - \mu(x) + O(1))$
    * Case 2: $3(\mu'(x) - \mu(x))$
    * Case 3: $3(\mu'(x) - \mu(x))$
  
  \[ \downarrow \]

  When we sum over all $\leq k + 1$ operations in a splay we get $3(\mu(T) - \mu(x) + O(1))$ where $\mu(x)$ is the number of credits on $x$ before the SPLAY.

  Note that it is important that we only have the $O(1)$ term in case 1.

• Case 1:

  – We have: $\mu'(x) = \mu(y), \mu'(y) \leq \mu'(x)$ and all other $\mu$'s are unchanged.
  – To maintain invariant we use: $\mu'(x) + \mu'(y) - \mu(x) - \mu(y) = \mu'(y) - \mu(x) \leq \mu'(x) - \mu(x) \leq 3(\mu'(x) - \mu(x))$

• Case 2:

  – We have $\mu'(x) = \mu(z), \mu'(y) \leq \mu'(x), \mu'(z) \leq \mu'(x), \mu(y) \geq \mu(x)$ and all other $\mu$'s are unchanged.
  – To maintain invariant we use:
    \[ \mu'(x) + \mu'(y) + \mu'(z) - \mu(x) - \mu(y) - \mu(z) = \mu'(y) + \mu'(z) - \mu(x) - \mu(y) = (\mu'(y) - \mu(x)) + (\mu'(z) - \mu(y)) \leq (\mu'(x) - \mu(x)) + (\mu'(x) - \mu(x)) = 2(\mu'(x) - \mu(x)) \]

  – This means that we can use the remaining $\mu'(x) - \mu(x)$ credits to pay for rotation, unless $\mu'(x) = \mu(x)$ (can happen since $\mu(x) = \lceil \log |T(x)| \rceil$).
  – We will show that if $\mu'(x) = \mu(x)$ then $\mu'(x) + \mu'(y) + \mu'(z) < \mu(x) + \mu(y) + \mu(z)$ which means that the operation actually releases credits we can use for the rotation:
    * Assume $\mu'(x) = \mu(x)$ and $\mu'(x) + \mu'(y) + \mu'(z) \geq \mu(x) + \mu(y) + \mu(z)$
    * We have $\mu(z) = \mu'(x) = \mu(x)$

  \[ \downarrow \]

  $\mu(z) = \mu(x) = \mu(y)$

  and $\mu'(x) + \mu'(y) + \mu'(z) \geq \mu(x) + \mu(y) + \mu(z) = 3\mu(x) = 3\mu'(x)$

  \[ \downarrow \]

  $\mu'(y) + \mu'(z) \geq 2\mu'(x)$

  * Since $\mu'(y) \leq \mu'(x)$ and $\mu'(z) \leq \mu'(x)$ we get $\mu'(x) = \mu'(y) = \mu'(z)$
  * Since $\mu(z) = \mu'(x)$ we have $\mu(x) = \mu(y) = \mu(z) = \mu'(x) = \mu'(y) = \mu'(z)$ which cannot be true (and thus our initial assumption cannot be true):
Let $a$ be $|T(x)|$ before rotations ($a = |T1| + |T2| + 1$)
Let $b$ be $|T(z)|$ after rotations ($b = |T3| + |T4| + 1$)
Since $\mu(x) = \mu'(z) = \mu'(x)$ we have $[\log a] = [\log b] = [\log(a + b + 1)]$ but then we have the following contradiction:

- if $a \leq b$: $[\log(a + b + 1)] \geq [\log 2a] = 1 + [\log a] > [\log a]$
- if $a > b$: $[\log(a + b + 1)] \geq [\log 2b] = 1 + [\log b] > [\log b]$

- Case 3:
  - Can be proved analogously to case 2.