Splay Trees

Splay Trees are Binary Search Trees (§ 9.3)

- BST Rules:
  - entries stored only at internal nodes
  - keys stored at nodes in the left subtree of \( v \) are less than or equal to the key stored at \( v \)
  - keys stored at nodes in the right subtree of \( v \) are greater than or equal to the key stored at \( v \)

- An inorder traversal will return the keys in order

Searching in a Splay Tree:
Starts the Same as in a BST

Search proceeds down the tree to found item or an external node.

Example: Search for time with key 11.

Example Searching in a BST, continued

search for key 8, ends at an internal node.
Splay Trees do Rotations after Every Operation (Even Search)

- new operation: splay
  - splaying moves a node to the root using rotations

- right rotation
  - makes the left child $x$ of a node $y$ into $y$'s parent; $y$ becomes the right child of $x$

- left rotation
  - makes the right child $y$ of a node $x$ into $x$’s parent; $x$ becomes the left child of $y$

(a) right rotation about $y$
(b) left rotation about $x$

(structure of tree above $y$ is not modified)
(structure of tree above $x$ is not modified)

Visualizing the Splaying Cases

Splaying:

- $x$ is a left-left grandchild means $x$ is a left child of its parent, which is itself a left child of its parent
- $p$ is $x$’s parent; $g$ is $p$’s parent

start with node $x$

1. is $x$ the root?
2. is $x$ a child of the root?
3. is $x$ the left child of the root?
4. right-rotate about the root
5. left-rotate about the root

Splaying Example

- let $x = (8,N)$
  - $x$ is the right child of its parent, which is the left child of the grandparent
  - left-rotate around $p$, then right-rotate around $g$
Splaying Example, Continued

1. (before applying rotation)
   - now \( x \) is the left child of the root
   - right-rotate around root

2. (after rotation)
   - \( x \) is the root, so stop

Example Result of Splaying

- A tree might not be more balanced
e.g. splay (40,X)
- before, the depth of the shallowest leaf is 3 and the deepest is 7
- after, the depth of the shallowest leaf is 1 and the deepest is 8

Splay Tree Definition

- a *splay tree* is a binary search tree where a node is splayed after it is accessed (for a search or update)
  - deepest internal node accessed is splayed
  - splaying costs \( O(h) \), where \( h \) is height of the tree
    - which is still \( O(n) \) worst-case
  - \( O(h) \) rotations, each of which is \( O(1) \)

Splay Trees & Ordered Dictionaries

- which nodes are splayed after each operation?
  - use the parent of the internal node that was actually removed from the tree (the parent of the node that the removed item was swapped with)

<table>
<thead>
<tr>
<th>method</th>
<th>splay node</th>
</tr>
</thead>
<tbody>
<tr>
<td>find(k)</td>
<td>if key found, use that node</td>
</tr>
<tr>
<td></td>
<td>if key not found, use parent of ending external node</td>
</tr>
<tr>
<td>insert(k,v)</td>
<td>use the new node containing the entry inserted</td>
</tr>
<tr>
<td>remove(k)</td>
<td>use the parent of the internal node that was actually removed from the tree (the parent of the node that the removed item was swapped with)</td>
</tr>
</tbody>
</table>
Amortized Analysis of Splay Trees

- Running time of each operation is proportional to time for splaying.
- Define rank(v) as the logarithm (base 2) of the number of nodes in subtree rooted at v.
- Costs: zig = $1, zig-æg = $2, zig æg = $2.
- Thus, cost for playing a node at depth d = $d.
- Imagine that we store rank(v) cyber-dollars at each node v of the splay tree (just for the sake of analysis).

Cost per zig

- Doing a zig at x costs at most rank'(x) - rank(x):
  \[ \text{cost} = \text{rank}'(x) + \text{rank}'(y) - \text{rank}(y) - \text{rank}(x) \]
  \[ \leq \text{rank}'(x) - \text{rank}(x). \]

Cost per zig-zig and zig-zag

- Doing a zig-æg or zig æg at x costs at most \[ 3(\text{rank}'(x) - \text{rank}(x)) - 2. \]

Cost of Splaying

- Cost of splaying a node x at depth d of a tree rooted at r:
  - at most \[ 3(\text{rank}(r) - \text{rank}(x)) - d + 2: \]
  - Proof: Splaying x takes \( \frac{d}{2} \) splaying substeps:
    \[ \text{cost} \leq \sum_{i=1}^{\frac{d}{2}} \text{cost}_i \]
    \[ \leq \sum_{i=1}^{\frac{d}{2}} (3(\text{rank}_{i-1}(x) - \text{rank}_{i-1}(x)) - 2) + 2 \]
    \[ = 3(\text{rank}(r) - \text{rank}_{i-1}(x)) - 2d / d + 2 \]
    \[ \leq 3(\text{rank}(r) - \text{rank}(x)) - d + 2. \]
Recall: rank of a node is logarithm of its size.
Thus, amortized cost of any splay operation is $O(\log n)$.
In fact, the analysis goes through for any reasonable definition of rank($x$).
This implies that splay trees can actually adapt to perform searches on frequently-requested items much faster than $O(\log n)$ in some cases. (See Proposition 9.4 and 9.5.)