Lecture 25: Balanced Binary Search Trees

CS 62
Spring 2019
William Devanny & Alexandra Papoutsaki
Rotating Binary Trees

Key idea: Rotate node higher in tree while keeping it in order.
Rotating Trees

• Rotate A to root (Right rotation)
  • All nodes in subtrees $\alpha$ go up one level, all in $\gamma$ go down one level, all in $\beta$ stay same.

• Rotate B to root (Left Rotation)
  • All nodes in subtrees $\gamma$ go up one level, all in $\alpha$ go down one level, all in $\beta$ stay same.

• See code in BinaryTree.java
Shifting elements toward root

- Move $x$ up two levels with two rotations
- If $x$ is left child of a left child...

Figure 14.5

Two of the rotation pairs used in the splaying operation. The other cases are mirror images of those shown here.

- If $x$ is the root, we are done.
- If $x$ is a left (or right) child of the root, rotate the tree to the right (or left) about the root.
- If $x$ is the left child of its parent $p$, which is, in turn, the left child of its grandparent $g$, rotate right about $g$, followed by a right rotation about $p$ (Figure 14.5a). A symmetric pair of rotations is possible if $x$ is a left child of a left child. After double rotation, continue splay of tree at $x$ with this new tree.
- If $x$ is the right child of $p$, which is the left child of $g$, we rotate left about $p$, then right about $g$ (Figure 14.5b). The method is similar if $x$ is the left child of a right child. Again, continue the splay at $x$ in the new tree.

After the splay has been completed, the node $x$ is located at the root of the tree. If node $x$ were to be immediately accessed again (a strong possibility), the tree is clearly optimized to handle this situation. It is not the case that the tree becomes more balanced (see Figure 14.5a). Clearly, if the tree is splayed at an extremal value, the tree is likely to be extremely unbalanced. An interesting feature, however, is that the depth of the nodes on the original path from $x$ to the root of the tree is, on average, halved. Since the average depth of these
Shifting elements toward root

• If x is a right child of a left child...

Symmetric if interchangeable left and right
Splay Trees

- Self-adjusting Binary Search Trees
- Fast access to elements accessed recently (locality)
- Every time contains, add or remove an element $x$, move it to the root by a series of rotations (called “splay”-ing $x$)

- Splay trees are on average balanced
  - But are not strictly balanced, unlike AVL/Red-Black trees
  - On average height is $O(\log n)$
  - Worst case height is $O(n)$
  - Amortized cost is $O(\log n)$ for all operations

- Popular: caches, garbage collectors, routing
Splay operations

• Zig
  • Zag
• Zig-zig
  • Zag-zag
• Zig-zag
  • Zag-zig
Zig or Zag: node is child of root

Zig (Right Rotation on (x,y))

Zag (Left Rotation on (x,y))
Zig-zig (Left Left case)

Node $x$ has both parent $P$ and grandparent $G$. Both $x$ and $P$ are the left children of their parents.
Zag-zag (Right Right case)

Node $x$ has both parent $P$ and grandparent $G$. Both $x$ and $P$ are the right children of their parents.
Zig-zag (Left Right case)

Node $x$ has both parent $P$ and grandparent $G$. $x$ is right child and $P$ is left child of their parents.
Zag-zig (Right Left case)

Node $x$ has both parent $P$ and grandparent $G$. $x$ is left child and $P$ is right child of their parents.
Operation sketches

- **contains**: Use locate method of BSTs. Splay the located node.
- **add**: Use add method of BSTs and then splay the new node.
- **remove**: Use contains to splay the node to be removed to the root. Delete it. You now have two independent trees. Find the maximum node in left subtree and splay it. Make root of right subtree its right child.

- [https://www.cs.usfca.edu/~galles/visualization/SplayTree.html](https://www.cs.usfca.edu/~galles/visualization/SplayTree.html)
Practice time

- add: 8 1 80 50 2 60 90 51
Practice time

- contains: 2
Practice time

- remove: 51
Static/Dynamic Optimality

Static Optimality Theorem

• The time it takes for a sequence of lookups in a splay tree is only a constant factor worse than ANY naïve binary search tree assuming every node is looked up once.

Dynamic Optimality Conjecture:

• The time it takes for a sequence of lookups in a splay tree is only a constant factor worse than ANY dynamic binary search tree assuming every node is looked up once. (AVL, Red-Black, others)