Rotations in Binary Search Trees

CLRS: Ch 13.2

- Since each of the basic bst operations takes time $O(h)$ where h is the tree height, efficient bst algorithms must find a way to keep the tree "balanced." Many balancing schemes have been discovered, e.g., weight-balanced, height-balanced, rank-balanced, and self-organization.

  We'll study the red-black trees and the splay trees. The former uses rank-balanced scheme; the latter uses self-organization.

- Rotation is the basic operation for efficient bst algorithms. A rotation rearranges subtrees but preserves inorder.

[EXAMPLE ROTATION PICTURE HERE]

A right (left) rotation can be performed at a node x if x has a left (right) child.

It's a misnomer to say we rotate at a node. It's better to think of rotating an edge. A left (right) rotation turns a right (left) edge into a left (right) edge.

A rotation changes no more than 3 child pointers & 3 parent pointers, so it can be done in time $O(1)$.

A rotation can change the height of the tree by one.

Theorem (CLRS Exercise 13.2-4)**.** *Let T_1 and T_2 be two bst's containing the same n nodes. There exists of sequence of ratations of length $\leq 2n-2$ that transforms T_1 into T_2.*

Proof. Use a right (or left) chain of nodes as an intermediate tree. □

- Efficient bst algorithms use a combination of rotations to rearrange the tree. It's essential to understand these patterns of rearrangement. They are best understood by thinking of them as rearrangement of a path of length two. There are 3 kinds of length-2 paths: *zig-zig*, *zig-zag*, and *sibling* paths.

[EXAMPLE PATH REARRANGEMENT PICTURE HERE]

Red-black tree algorithms rearrange both zig-zig and zig-zag paths into a sibling path. Splay tree algorithms rearrange a zig-zag path into a sibling path, and rearrange a zig-zig path into another zig-zig path.

Each rearrangement is accomplished by performing at most two rotations in a row.