伸展树学习文档（备忘录）

简化自一个ppt

Thoughts

- Balanced BSTs aren't necessarily optimal!Goal: Construct a binary search tree T* that minimizes the total expected access time.
- Challenge: Can we construct an optimal BST without knowing the access probabilities in advance?
- After looking up an element, repeatedly rotate that element with its parent until it becomes the root.
- Problem: Rotating an element x to the root significantly “helps” x, but “hurts” the rest of the tree.Most of the nodes on the access path to xhave depth that increases or is unchanged.
- Splay: Rotates an element to the root of the tree, but does so in a way that's more “fair” to other nodes in the tree.

Splay

3 cases：zigzig zigzag zig

Why Splaying works?

Claim: After doing a splay at x, the average depth of any nodes on the access path to x is halved.Intuitively, splaying x benefits nodes near x, not just x itself.This “altruism” will ensure that splays are efficient. Each rotation done only slightly penalizes each other part of the tree (say, adding +1 or +2 depth).But Each splay rapidly cuts down the height of each node on the access path.

Lookups Operations

To do a lookup in a splay tree: ● Search for that item as usual. ● If it's found, splay it up to the root. ● Otherwise, splay the last-visited node to the root.

Insertion&Delete Operations

To insert a node into a splay tree: ● Insert the node as usual. ● Splay it up to the root. To delete a key k from the tree: ● Splay k to the root. ● Delete k. ● Join the two resulting subtrees.

Join&Split Operations

To join two trees T_1 and T_2, where all keys in T_1 are less than the keys in T_2: ● Splay the max element of T_1 to the root. ● Make T_2 a right child of T_1. To split T at a key k: ● Splay the successor of k up to the root. ● Cut the link from the root to its left child.

All of these operations require amortized time O(log n).

Compared with RedBlackTree

● No need to store any kind of balance information. ● Only three rules to memorize.