Lecture 19: Red-black tree (13.1-13.2)

CS303: Algorithms

Last update: February 22, 2014

1 Review

- BST's properties: advantage and disadvantage comparing with hashing
- Operations on BST: search, min, max, predecessor, successor (overall efficiency is almost all $\Theta(h)$ where n is the number of nodes in the BST

2 Rationale of Red-black tree

BST is useful with pretty good efficiency on dictionary operations if h is lgn. However, if the tree is heavily skewed, then h can be as large as n which isn't good.

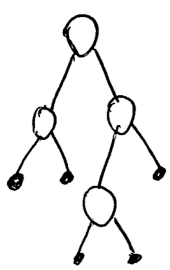
Q:/ Solution?

A:/ balanced binary search tree. Red-black tree is one of them.

Red-black tree is a balanced binary search tree structure which maintains dynamic set of n elements using tree of height $O(lgn)$

Idea: add an extra field called "color field" to each node satisfying the following properties

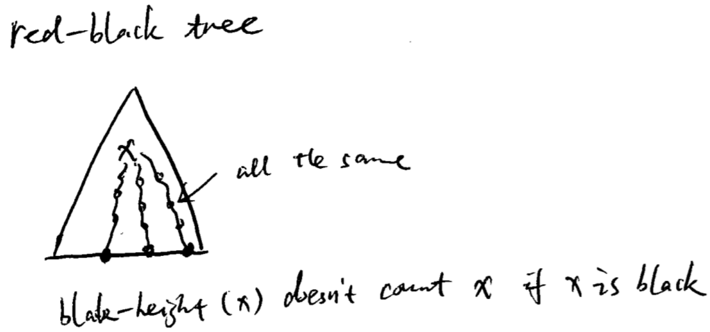
- every node is either red or black. We will use double circle for red and circle for black
- The root and leaves are black. We will basically add nils as children to all nodes with less than 2 children.



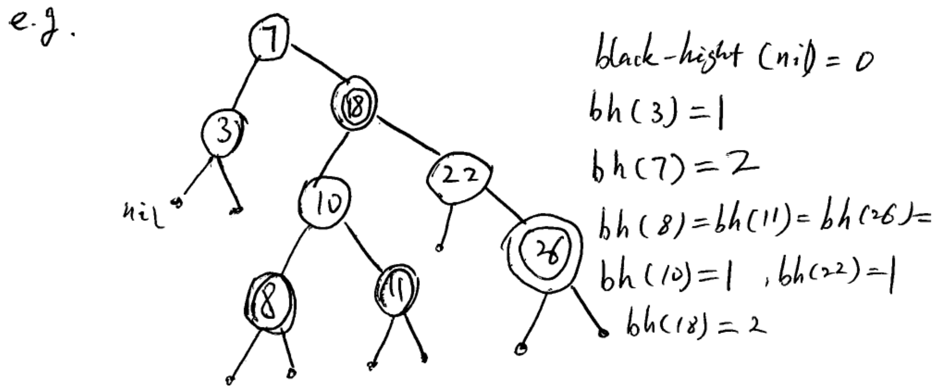
the • are nils we'll add so that all internal nodes have 2 children. All leaves are nils (black)

- Every red node has black parent. → you can never have two consecutive red nodes in a path.

- All simple paths from a node x to a descendant leaf of x have the same number of black nodes on them. Denote this as $black - height(x)$



The purpose is to force the height of this kind of BST to be $O(lgn)$. In fact, if we make every node in a regular BST black and add in the nils, the first three conditions will be satisfied.

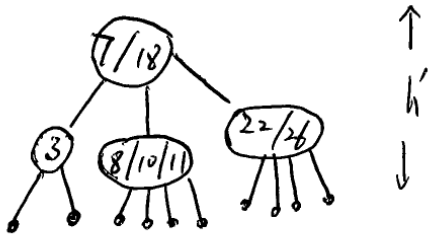


3 Topic 3: Height of red-black tree

3.1 Red-black tree with n keys has height$h \leq 2lg(n+1)$. Note that n keys don't include the nils we add.

(sketchy) proof: I will first merge each red node into its parent (note that the parent of a red node must be black)

- every internal node has 2 or 3 or 4 children (it is called 2-3-4 tree)
- All the leaves have the same depth, namely black-height(root)



Note: now all the leaves have the same depth because we basically ignored all the red nodes.

Let h' be the height of the 2-3-4 tree and h be the height of red-black tree

- The number of leaves in both trees are $n + 1$ because all internal children have a branching factor of 2.
- For the 2-3-4 tree, the number of leaves is at least $2^{h'}$ and at most $4^{h'} \leq n + 1$. Thus $h' <= lg(n + 1)$.
- In the merging process, we at most insert a red node into a black parent every other black node. Thus $h <= 2h'$

Thus the property of $h \leq 2lg(n + 1)$ is proved.

Corollary: Queries (search, min, max, successor, predecessor) are all $O(lgn)$ time in a red-black tree.

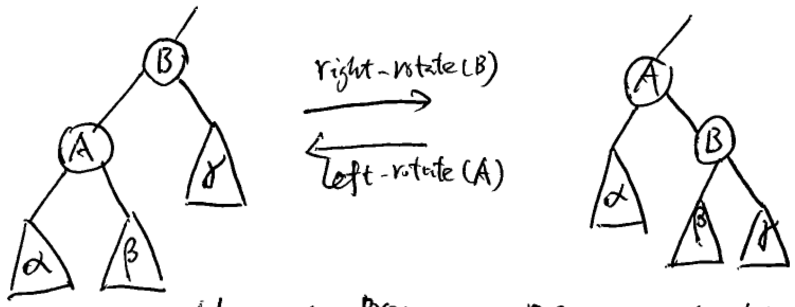
4 Updates on red-black tree (insertion and deletion)

We have to maintain the properties of the red-black tree.

- BST operation
- color change (recolor step)
- restructuring of links via rotations

Rotation:

Rotations



this op preserves BST property!

this op also takes constant time because we only change a constant time change of pointers