**CS430 Lecture 11 Activities**

Opening Questions

1. What is the main problem with Binary Search Trees that Red-Black Trees correct? Explain briefly (2-3 sentences) how Red-Black Trees correct this problem with Binary Search Trees.

2. For the balanced binary search trees, why is it important that we can show that a rotation at a node in a is O(1) (i.e. not dependent on the size the BST)

Red-Black Trees  
Red-Black Properties

1. Every node is colored either red or black
2. The root is black
3. Every null pointer descending from a leaf is considered to be a null black leaf node
4. If a node is red, then both its children are black
5. For each node, all paths from the node to descendant leaves contain the same number of black nodes (black height)

black-height of a node bh(x) - The number of black nodes on any path from, but not including, a node "x" down to a null black leaf node

1. If a node x has bh(x)=3, what is its largest and smallest possible height (distance to farthest leaf) in the BST?

2. Prove using induction and red-black tree properties. A red-black tree with n internal nodes (n key values) has height at most 2lg(n+1)

Part A - First show the sub-tree rooted at node "x " has at least 2bh(x)-1 internal nodes. Use induction

Part B – Let “h” be height of R-B Tree, by property 4 at least half the nodes on path from root to leaf are black

bh(root) >= h/2

Use that and part A to show h <= 2log(n+1)

3. Which BST operations change for a red-black tree and which do not change? What do the operations that change need to be aware of and why?

search, insert, delete, predecessor, successor, minimum, maximum, rotations

Red-Black Tree Insert - Similar to BST insert, assume we start with a valid red-black tree.

1. Locate leaf position to insert new node
2. Color new node red and create 2 new black nil leafs below newly inserted red node
3. If parent of new insert was \_\_\_\_\_\_\_\_\_\_\_(fill in the blank, black or red), then done. ELSE procedure to recolor nodes and perform rotations to maintain red-black properties.

There are three cases if R-B Property #4 broken when insert a red node "Z" (or changed color of a node to red) and its parent is also red.

|  |  |
| --- | --- |
|  | Case 1. Node "Z" (red) is a left or right child and its parent is red and its uncle is red (the children of nodes value 4 5 8 must all be black, or nil black). Case 1 swap the colors of a parent node and both its children, preserving the black height property at all nodes.   * Change Z’s parent and uncle to black * Change Z’s grandparent to red * No effect on black height on any node * Z’s grandparent is now Z and check again for property #4 (two reds in a row) still broken at new node Z (possible non-terminal case, need loop or recursion) |
|  | Case 2. Node "Z" is a right child and its parent is red and its uncle is NOT red. Case 2 does a single rotation, preserving the black height property at all nodes.   * Rotate left on parent of Z * Re-label old parent of Z as Z and continue to case #3 |
|  | Case 3. Node "Z" is a left child and its parent is red and its uncle is NOT red. Case 3 does a single rotation and swaps the colors of a parent node and both its children, preserving the black height property at all nodes.   * Rotate right on grandparent of Z * Color old parent of Z black * Color old grandparent of Z red |