Data Structures and Algorithms

AVL Tree Insertion

Printable Version

Start out by using a regular binary search tree insertion. Set the balance of the newly inserted node (it should be zero left height, zero right height, and height of 1). Call *insertionFixUp*, passing a pointer to the newly inserted node, *x*.

```
function insertionFixup(x)
    {
    loop
        {
        if (x is the root)
            exit the loop
        else if (parent favors the sibling)        //case 1
            {
            set the balance of parent
            exit the loop
            }
        else if (parent has no favorite)           //case 2
            {
            set the balance of parent
            x = parent
            //continue looping
            }
        else //parent favors x
            {
            y = the favorite child of x
            p = parent of x
            if (y exists and y,x,p are not linear) //case 3
                {
                rotate y to x
                rotate y to p
                set the balance of x
                set the balance of p
                set the balance of y
                }
            else                                   //case 4
                {
                rotate x to p
                set the balance of p
                set the balance of x
                }
            exit the loop
            }
        }
    }
```

Setting the balance of a node modifies three fields, the *height*, the *leftHeight*, and the

rightHeight. The last two fields cache the heights of the left and right children, respectively:

```
function setBalance(x)
    {
    set x's leftHeight to the height of the left child,
        zero if there is no left child
    set x's rightHeight to the height of the right child,
        zero if there is no right child
    set x's height to the max of leftHeight and rightHeight, plus one
    }
```

Note that in this pseudocode, there are no references to leftness and rightness. This issue is deferred to the helper functions. For example, the getting the sibling of a node *c* with parent *p* could be implemented as:

```
function sibling(c)
    {
    if c is its parent's left child, return the parent's right child
    if c is its parent's right child, return the parent's left child
    }
```

Next: Deleting from *AVL* trees