More Red-Black Trees

10-14-2003

Opening Discussion

- What did we talk about last class?
- Do you have any questions about the assignment?
- Do you have any questions about the midterm?
- The value of balanced binary trees.

Revisiting Insertion

```
void InsertFixup(Node z) {
        while(z.parent.color==RED) {
                if(z.parent==z.parent.parent.left) {
                        y=z.parent.parent.right;
                        if(y.color==RED) {
                                z.parent.color=BLACK;
                                y.color=BLACK;
                                z.parent.parent.color=RED;
                                z=z.parent.parent;
                        } else {
                                if(z==z.parent.right) {
                                        z=z.parent;
                                        RotateLeft(z);
                                }
                                z.parent.color=BLACK;
                                z.parent.parent.color=RED;
                                RotateRight(z.parent.parent);
                        }
                } else { same as above with left and right switched }
        }
        root.color=BLACK;
}
```

Deletion

- The delete your book uses is a bit different from what we did. The main difference is that they don't actually move the node we are deleting. Instead, they change the value stored in it. This has less impact on the red-black structure, though we can make ours look like that by remembering to set the color in the replacement node to what it is replacing.

Deletion Code

```
void Delete(Node z) {
      if(z.left==0 || z.right==0) y=z;
      else y=Successor(z);
      if(y.left!=0) x=y.left;
      else x=y.right;
      x.parent=y.parent;
      if(y.parent==0) {
            root=x;
      } else {
            if(y==y.parent.left) y.parent.left=x;
            else y.parent.right=x;
      }
      if(y!=z) {
            z.data=y.data;
      }
      if(y.color==BLACK) {
            DeleteFixup(x);
      }
}
```

Deletion Fixing

- After the delete we again have to fix things, but only if the sliced out node was black. This function is distinctly different from the fixing that is used in insertion. In this regard, the code can be more complex than the AVL tree.

Deletion Fixing Loop

```
while(x!=root && x.color==BLACK) {
       if(x==x.parent.left) {
              w=x.parent.right;
              if(w.color==RED) {
                     w.color=BLACK; x.parent.color=RED;
                     RotateLeft(x.parent); w=x.parent.right;
              }
              if(w.left.color==BLACK && w.right.color==BLACK) {
                     w.color=red; x=x.parent;
              } else {
                     if(w.right.color==BLACK){
                            w.left.color=BLACK; w.color=RED;
                            RotateRight(w); w=x.parent.right;
                     }
                     w.color=x.parent.color; x.parent.color=BLACK;
                     w.right.color=BLACK; RotateLeft(x.parent);
                     x=root;
              }
       } else { same as above but exchange right and left }
}
x.color=black;
```

Code

- Let's continue to work on our binary tree code so that we feel confident that we have a working, non self-balancing binary tree.

Minute Essay

- Do you think you will have assignment #2 completed by tonight? If not, what parts of it are hanging you up?
- We have the midterm next class. We'll have a review session for it tomorrow at 4:00 in 228. I stay until 6pm or whenever the questions run out. It's intermingled with ACM programming team practice after 4:30. Those who can't make that can meet right now for a bit.