# Red–Black Tree Notes – Handout

Computer Science 223p
Python Programming

A red–black tree is a binary search tree with one extra bit of storage per node which denotes the node's color. A node can be *red* or *black*. By labeling the nodes with a color and following a few simple rules that will be detailed, on any simple path from the root to a leaf, the red–black tree ensures that no such path is more than twice as long as any other. This means that the tree is approximately balanced.

A red–black tree satisfies the following properties.

- Every node is either red or black.
- The root is black.
- Ever leaf is black. (NIL nodes count as leaf nodes.)
- If a node is red, then both its children are black.
- For each node, all simple paths form the node to descendant leaves contain the same number of black nodes.

A good search tree is a balanced search tree. Balanced means that the left subtree is of the same height, or close to the same height, as the right subtree. The properties of a red–black tree mean that we can expect a red–black tree with $n$ internal nodes to have a height of at most $2\lg(n+1)$.

The code listings in this handout are taken from the identified pages of Introduction to Algorithms, 3rd Ed. by Cormen et al. This is an excellent reference book and all computer science students are encouraged to acquire their own copy.

```
// p. 313
Left-Rotate(T, x)
  y = x.right
  x.right = y.left
  if y.left ≠ T.nil
    y.left.p = x
  y.p = x.p
  if x.p == T.nil
    T.root = y
  else if x == x.p.left
    x.p.left = y
  else
    x.p.right = y
  y.left = x
  x.p = y

Right-Rotate(T, x)
  // A mirror image of Left-Rotate

// p. 315
RB-Insert(T, z)
  y = T.nil
  x = T.root
  while x ≠ T.nil
    y = x
    if z.key < x.key
      x = x.left
    else
      x = x.right
```

```
  z.p = y
  if y == T.nil
    T.root = z
  else if z.key < y.key
    y.left = z
  else
    y.right = z
  z.left = T.nil
  z.right = T.nil
  z.color = RED
  Insert-Fixup(T, z)

// p. 316
RB-Insert-Fixup(T, z)
  while z.p.color == RED
    if z.p == z.p.p.left
      y = z.p.p.right
      if y.color == RED
        z.p.color = BLACK
        y.color = BLACK
        z.p.p.color = RED
        z = z.p.p
      else
        if z == z.p.right
          z = z.p
          Left-Rotate(T, z)
        z.p.color = BLACK
        z.p.p.color = RED
        Right-Rotate(T, z.p.p)
    else
      // Same as above but reflected
  T.root.color = BLACK

// p. 323
RB-Transplant(T, u, v)
  if u.p == T.nil
    T.root = v
  else if u == u.p.left
    u.p.left = v
  else
    u.p.right = v
  v.p = u.p

// p.324
RB-Delete(T, z)
  y = z
  y-original-color = y.color
  if z.left == T.nil
    x = z.right
    Transplant(T, z, z.right)
  else if z.right == T.nil
```

```
    x = z.left
    Transplant(T, z, z.left)
  else
    y = Minimum(z.right)
    y-original-color = y.color
    x = y.right
    if y.p == z
      x.p = y
    else
      Transplant(T, y, y.right)
      y.right = z.right
      y.right.p = y
    Transplant(T, z, y)
    y.left = z.left
    y.left.p = y
    y.color = z.color
  if y-original-color == BLACK
    Delete-Fixup(T, x)

// p. 326
RB-Delete-Fixup(T, x)
  while x ≠ T.root and x.color == BLACK
    if x == x.p.left
      w = x.p.right
      if w.color == RED
        w.color = BLACK
        x.p.color = RED
        Left-Rotate(T, x.p)
        w = x.p.right
      if w.left.color == BLACK and w.right.color == BLACK
        w.color = RED
        x = x.p
      else
        if w.right.color == BLACK
          w.left.color = BLACK
          w.color = RED
          Right-Rotate(T, w)
          w = x.p.right
        w.color = x.p.color
        x.p.color = BLACK
        w.right.color = BLACK
        Left-Rotate(T, x.p)
        x = T.root
    else
      // Same as above but reflected
  x.color = BLACK
```

$Id: redblack_tree_algorithms.tex 4345 2013-08-24 03:15:45Z mshafae $