Operations on RB Tree :

Initializing the RB tree

ngx_rbtree_init(tree,s,i)

Source code :

```
#define ngx_rbtree_init(tree, s, i)                                     \
    ngx_rbtree_sentinel_init(s);                                        \
    (tree)->root = s;                                                   \
    (tree)->sentinel = s;                                               \
    (tree)->insert = i
```

This macro is used to initialize the rbtree instance.

'tree' is the pointer to the 'rbtree' instance of type ngx_rbtree_t.

's' is the pointer to the sentinel node

'i' is pointer to the insert function

Inserting into RB tree :

This function is called by application to put a new node in the rbtree. This function internally calls the insert function provided to ngx_rbtree_init . Then after insertion this function rebalances the tree at every insertion.

```
void ngx_rbtree_insert(ngx_thread_volatile ngx_rbtree_t *tree, ngx_rbtree_node_t *node)
{
    ngx_rbtree_node_t  **root, *temp, *sentinel;
    root = (ngx_rbtree_node_t **) &tree->root;
    sentinel = tree->sentinel;
    if (*root == sentinel) {
        node->parent = NULL;
        node->left = sentinel;
        node->right = sentinel;
        ngx_rbt_black(node);
        *root = node;
        return;
    }
    tree->insert(*root, node, sentinel);
    /* re-balance tree */
```

```
while (node != *root && ngx_rbt_is_red(node->parent)) {
    if (node->parent == node->parent->parent->left) {
        temp = node->parent->parent->right;

        if (ngx_rbt_is_red(temp)) {
            ngx_rbt_black(node->parent);
            ngx_rbt_black(temp);
            ngx_rbt_red(node->parent->parent);
            node = node->parent->parent;
        } else {
            if (node == node->parent->right) {
                node = node->parent;
                ngx_rbtree_left_rotate(root, sentinel, node);
            }
            ngx_rbt_black(node->parent);
            ngx_rbt_red(node->parent->parent);
            ngx_rbtree_right_rotate(root, sentinel, node->parent->parent);
        }
    } else {
        temp = node->parent->parent->left;
        if (ngx_rbt_is_red(temp)) {
            ngx_rbt_black(node->parent);
            ngx_rbt_black(temp);
            ngx_rbt_red(node->parent->parent);
            node = node->parent->parent;
        } else {
            if (node == node->parent->left) {
                node = node->parent;
                ngx_rbtree_right_rotate(root, sentinel, node);
            }
            ngx_rbt_black(node->parent);
            ngx_rbt_red(node->parent->parent);
            ngx_rbtree_left_rotate(root, sentinel, node->parent->parent);
        }
    }
}
```

```
    ngx_rbt_black(*root);
}
```

Deleting node in RB tree :

void ngx_rbtree_delete(ngx_thread_volatile ngx_rbtree_t *tree, ngx_rbtree_node_t *node)

Parameters :

tree : the pointer to the rbtree structure from which node is to be deleted.

node : The pointer to the node to be deleted.

This function is called to remove ‘node’ from the rbtree ‘tree’.It removes the node from rb tree and rebalances the tree.