Nginx Red Black Tree

Table of contents :

Introduction :

The concept of RB Tree is implemented in nginx using these two files :

- ngx_rbtree.c -> Present in src/core/ directory
- ngx_rbtree.h -> Present in src/core/ directory

In nginx RB tree is used to store open file cache entries and to cache SSL sessions across threads.The data which is best suited to be stored in tree structure , at those situations RB trees should be used.

Red Black tree provides improvement over the binary trees as RB Trees rebalances itself and thus the height of the tree remains constant unlike the ordinary binary tree.

Concept of RB Tree :

Basic binary search trees are simple data structures that have O(log N) search, insertion, and deletion. For the most part this is true, assuming that the data arrives in random order, but basic binary search trees degrade in cases when the input is ascending or descending sorted order (the third is outside-in alternating order). Because binary search trees store their data in such a way that can be considered sorted, if the data arrives already sorted, this causes problems. Consider adding the values 0,1,2,3,4 to a binary search tree. Since each new item is greater in value than the last, it will be linked to the right subtree of every item before it:

```
 0
  \
   1
    \
     2
      \
       3
        \
         4
```

The performance degrades from O(log N) to O(N) because the tree is now effectively a linear data structure.

Thus RB trees are used as they rebalance themselves and provide a constant O(log N) time for search, insertion, and deletion.

RB Tree Structure :

The RB tree Structure in nginx is defined as :

```
typedef struct ngx_rbtree_s  ngx_rbtree_t;

struct ngx_rbtree_s {
    ngx_rbtree_node_t     *root;
    ngx_rbtree_node_t     *sentinel;
    ngx_rbtree_insert_pt   insert;
};
```

Here :

root : This is the root node of the RB Tree
sentinel : This is empty node.The sentinel node is an empty node.It is an alternative to NULL pointer.Instead of making all the leaf nodes point to NULL separately,we can make them point to a single sentinel node.This node doesn’t store anything.The advantage of using this sentinel node over NULL is that we don’t have to check for null value each time we want to access the node thus increasing the efficiency.
insert : Insert is a function pointer which points to the insert function which is to be used to insert the nodes in RB Tree.
From the above structure we can see that RB Tree does’nt provide all the binary tree operations.In nginx API the insertion operation on RB Tree is left to the users to implement.This reason for doing this is that nginx doesn’t know what type of value we are going insert.It can be number,string or anything else.But,nginx performs the rebalancing of the tree.Nginx also provides deletion and searching operations on RB Tree.

Here We see that Root,Sentinel, are all nodes of a RB Tree. The node structure of the RB Tree is defined as follows :

```
typedef struct ngx_rbtree_node_s  ngx_rbtree_node_t;

struct ngx_rbtree_node_s {
    ngx_rbtree_key_t       key;
    ngx_rbtree_node_t     *left;
    ngx_rbtree_node_t     *right;
    ngx_rbtree_node_t     *parent;
    u_char                 color;
    u_char                 data;
};
```

Here ,
key : The hash value of the rbtree node data is put in this field.This key is compared to check whether to put the incoming node on left or right side.The main purpose of using key comparison instead od data comparison is that key comparison is actually a numeric comparison which is much faster than string comparison.
left : The pointer to the left child of the node
right : pointer to the right child of the node
parent : pointer to the parent of the node
color : The color of the node.
    0 – Signifies color is Black
    1 – Signifies color is red.
data : start of the data

To perform basic operations on RB Tree the following macros are defined.

```
#define ngx_rbt_red(node)               ((node)->color = 1)
```
This sets the color of the node to 1 (red)

```
#define ngx_rbt_black(node)             ((node)->color = 0)
```
This sets the color of the node to 1 (Black)

```
#define ngx_rbt_is_red(node)            ((node)->color)
```
This checks whether the node is red or not

```
#define ngx_rbt_is_black(node)          (!ngx_rbt_is_red(node))
```
This checks whether the node is Black or not

```
#define ngx_rbt_copy_color(n1, n2)      (n1->color = n2->color)
```
This copies the color of one node to other