```
#include <linux/rbtree.h>
```

data node in an rbtree containing a struct rb_node member

```
struct mytype {
    char *keystring;
    struct rb_node node;
};
```

- initialize the root of rbtree

```
struct rb_root mytree = RB_ROOT;
```

- write a search function

```
/*
 *
 * search the node that contains string
 *
 */
struct mytype *my_search(struct rb_root *root, char *string)

{

    struct rb_node *node = root->rb_node;

    while(node) {

        struct mytype *data = container_of(node, struct mytype, node);

        int result;

        result = strcmp(string, data->keystring);

        if (result < 0)

        node = node->rb_left;
        else if (result > 0)
        node = node->rb_right;
    else
        return  data;
        }
    return  NULL;
}
```

- inseert data into an rbtree
  step 1: searching for the place to insert the new node
  step 2: inserting the node and rebalancing the tree

```
int my_insert(struct rb_root *root, struct mytype *data)
{
    struct rb_node **new = &(root->rb_node), *parent = NULL;

    // step 1
    while (*new) {
        struct mytype *this = container_of(*new, struct mytype, node);
        int result = strcmp(data->keystring, this->keystring);
        parent = *new;
        if (result < 0)
            new = &((*new)->rb_left);
        else if (result > 0)
            new = &((*new)->rb_right);
        else
            return FALSE;
        }


    // step 2
    rb_link_node(data->node, parent, new);
    // reblance rbtree
    rb_insert_color(data->node, root);
        return TRUE;
}
```

- remove an existing data from an rbtree

  > void rb_erase(struct rb_node *victim, struct rb_root* tree);

```
struct mytype *data = mysearch(mytree, "walrus");
if (data)
{
    rb_erase(data->node, mytree);
    myfree(data);
}
```

- replace wn existing node in the rbtree with a new one with the same key

  > void rb_replace_node(struct rb_node *old, struct rb_node* new, struct rb_root *tree);

这里的关键是 `same key` ,如果key是不同的，则会corrupt整个rbtree.
这里的key就是struct mytype中的keystring.

如果key不同，则要先删除，然后再插入！

```
struct rb_node *rb_first(struct rb_root *tree);
struct rb_node *rb_last(struct rb_root *tree);
struct rb_node *rb_next(struct rb_node *node);
struct rb_node *rb_prev(struct rb_node *node);
```

这 4 个APIs用于traverse rbtree.
rb_first() return a pointer to the first node

rb_last() return a pointer to the last node

rb_next() return the next node

rb_prev() return the previous node

```c
struct rb_node *node;
for (node = rb_first(&mytree); node; node = rb_next(node))
    printk("key=%s\n", rb_entry(node, int, keystring));
```