

# TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

12회차 (2018-03-09)

강사 - Innova Lee(이상훈)

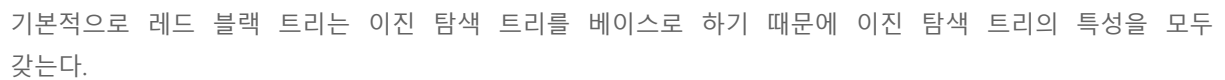
[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)

학생 - 정유경

[ucong@naver.com](mailto:ucong@naver.com)

<https://www.cs.usfca.edu/~galles/visualization/RedBlack.html>

© 2004 Blackwell Publishing Ltd *Journal of Internal Medicine* 255: 105–112



- ### 3. RB Tree의 성능

일반적인 경우에는 보다 타이트한 균형을 가지는 AVL 트리가 평균적인 탐색 시간은 더 빠르다.

삽입/삭제의 동작은 RB 트리가 우위이고 최악의 경우에 AVL 트리보다 탐색이 빠르다고 한다면 AVL 트리보다 RB 트리가 전체적으로 우위에 있다고 할 수 있다.

#### 4. RE Tree Source Code

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define BLACK 0
#define RED 1

typedef struct __rb_node
{
    int data;
    int color;

    struct __rb_node *left;
    struct __rb_node *right;
    struct __rb_node *parent;
} rb_node;

typedef struct __rb_tree
{
    struct __rb_node *root;
    struct __rb_node *nil;
} rb_tree;

bool is_dup(int *arr, int cur_idx)
{
    int i, tmp = arr[cur_idx];
    for (i = 0; i < cur_idx; i++)
        if (tmp == arr[i])
            return true;
    return false;
}

void init_rand_arr(int *arr, int size)
{
    int i;
    for (i = 0; i < size; i++)
    {
        redo:
            arr[i] = rand() % 200 + 1;

            if (is_dup(arr, i))
            {
                printf("%d dup! redo rand()\n", arr[i]);
                goto redo;
            }
    }
}

void rb_left_rotate(rb_tree **tree, rb_node *x)
{
    rb_node *y;
    rb_node *nil = (*tree)->nil;
```

```

    y = x->right;
    x->right = y->left;

    if (y->left != nil)
        y->left->parent = x;
    y->parent = x->parent;

    if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;

    y->left = x;
    x->parent = y;
}

void rb_right_rotate(rb_tree **tree, rb_node *y)
{
    rb_node *x;
    rb_node *nil = (*tree)->nil;

    x = y->left;
    y->left = x->right;

    if (nil != x->right)
        x->right->parent = y;

    x->parent = y->parent;
    if (y->parent->left == y)
        y->parent->left = x;
    else
        y->parent->right = x;

    x->right = y;
    y->parent = x;
}

void rb_tree_ins_helper(rb_tree **tree, rb_node *z)
{
    rb_node *x;
    rb_node *y;
    rb_node *nil = (*tree)->nil;

    z->left = z->right = nil;
    y = (*tree)->root;
    x = (*tree)->root->left;

    while (x != nil)
    {
        y = x;

        if (x->data > z->data)
            x = x->left;
        else
            x = x->right;
    }

```

```

    z->parent = y;

    if ((*tree)->root == y) || (y->data > z->data))
        y->left = z;
    else
        y->right = z;
}

rb_node *rb_tree_ins(rb_tree **tree, int data)
{
    rb_node *x;
    rb_node *y;
    rb_node *tmp;

    x = (rb_node *)malloc(sizeof(rb_node));
    x->data = data;
    rb_tree_ins_helper(tree, x);
    tmp = x;
    x->color = RED;

    while (x->parent->color)
    {
        if (x->parent == x->parent->parent->left)
        {
            y = x->parent->parent->right;

            if (y->color)
            {
                {
                    x->parent->color = BLACK;
                    y->color = BLACK;
                    x->parent->parent->color = RED;
                    x = x->parent->parent;
                }

            else
            {
                if (x->parent->right == x)
                {
                    {
                        x = x->parent;
                        rb_left_rotate(tree, x);
                    }
                    x->parent->color = BLACK;
                    x->parent->parent->color = RED;
                    rb_right_rotate(tree, x->parent->parent);
                }

            }

        else
        {
            y = x->parent->parent->left;

            if (y->color)
            {
                {
                    x->parent->color = BLACK;
                    y->color = BLACK;
                    x->parent->parent->color = RED;
                    x = x->parent->parent;
                }
            }
        }
    }
}

```

```

        }
        else
        {
            if (x->parent->left == x)
            {
                x = x->parent;
                rb_right_rotate(tree, x);
            }
            x->parent->color = BLACK;
            x->parent->parent->color = RED;
            rb_left_rotate(tree, x->parent->parent);
        }
    }

    (*tree)->root->left->color = BLACK;
    return tmp;
}

rb_tree *rb_tree_create(void)
{
    rb_tree *rbt;
    rb_node *tmp;

    rbt = (rb_tree *)malloc(sizeof(rb_tree));
    tmp = rbt->nil = (rb_node *)malloc(sizeof(rb_node));
    tmp->parent = tmp->left = tmp->right = tmp;
    tmp->color = BLACK;
    tmp->data = 0;
    tmp = rbt->root = (rb_node *)malloc(sizeof(rb_node));
    tmp->parent = tmp->left = tmp->right = rbt->nil;
    tmp->color = BLACK;
    tmp->data = 0;

    return rbt;
}

void rb_tree_preorder_print(rb_tree *tree, rb_node *x)
{
    rb_node *nil = tree->nil;
    rb_node *root = tree->root;

    if (x != tree->nil)
    {
        printf("data = %4i, ", x->data);

        if (x->left == nil)
            printf("left = NULL, ");
    }
}

```

```

        else
            printf("left = %4i, ", x->left->data);

        if (x->right == nil)
            printf("right = NULL, ");
        else
            printf("right = %4i, ", x->right->data);
        printf("color = %4i\n", x->color);
        rb_tree_preorder_print(tree, x->left);
        rb_tree_preorder_print(tree, x->right);
    }
}

void rb_tree_print(rb_tree *tree)
{
    rb_tree_preorder_print(tree, tree->root->left);
}

int data_test(int n1, int n2)
{
    if (n1 > n2)
        return 1;
    else if (n1 < n2)
        return -1;
    else
        return 0;
}

rb_node *rb_tree_find(rb_tree *tree, int data)
{
    int tmp;
    rb_node *x = tree->root->left;
    rb_node *nil = tree->nil;

    if (x == nil)
        return 0;

    tmp = data_test(x->data, data);

    while (tmp != 0)
    {
        if (x->data > data)
            x = x->left;

        else
            x = x->right;
        if (x == nil)
            return 0;
    }
}

```

```

        tmp = data_test(x->data, data);
    }

    return x;
}

rb_node *rb_tree_successor(rb_tree *tree, rb_node *x)
{
    rb_node *y;
    rb_node *nil = tree->nil;
    rb_node *root = tree->root;

    if (nil != (y = x->right))
    {
        while (y->left != nil)
            y = y->left;
        return y;
    }

    else
    {
        y = x->parent;

        while (y->right == x)
        {
            x = y;
            y = y->parent;
        }

        if (y == root)
            return nil;
        return y;
    }
}

void rb_tree_del_fixup(rb_tree *tree, rb_node *x)
{
    rb_node *root = tree->root->left;
    rb_node *w;

    while ((!x->color) && (root != x))
    {

```



```

if (x->parent->left == x)
{
    w = x->parent->right;

    if (w->color)
    {

        w->color = BLACK;

        x->parent->color = RED;

        rb_left_rotate(&tree, x->parent);

        w = x->parent->right;

    }

    if ((!w->right->color) && (!w->left->color))
    {

        w->color = RED;

        x = x->parent;

    }

    else
    {

        if (!w->right->color)
        {

            w->left->color = BLACK;

            w->color = RED;

            rb_right_rotate(&tree, w);

            w = x->parent->right;

        }

        w->color = x->parent->color;

        x->parent->color = BLACK;

        w->right->color = BLACK;

        rb_right_rotate(&tree, x->parent);

        x = root;

    }

}

else

```

```

        {
            w = x->parent->left;

            if (w->color)
            {
                w->color = BLACK;
                x->parent->color = 1;
                rb_right_rotate(&tree, x->parent);
                w = x->parent->left;
            }

            if ((!w->right->color) && (!w->left->color))
            {
                w->color = RED;
                x = x->parent;
            }
            else
            {
                if ((!w->right->color) && (!w->left->color))
                {
                    w->right->color = BLACK;

                    w->color = RED;
                    rb_left_rotate(&tree, w);
                    w = x->parent->left;
                }
                w->color = x->parent->color;
                x->parent->color = BLACK;
                w->left->color = BLACK;

                rb_right_rotate(&tree, x->parent);

                x = root;
            }
        }

    }
    x->color = BLACK;
}

void rb_tree_del(rb_tree *tree, rb_node *z)
{
    rb_node *y;
    rb_node *x;
    rb_node *nil = tree->nil;
    rb_node *root = tree->root;

    y = ((z->left == nil) || (z->right == nil)) ?
        z : rb_tree_successor(tree, z);
    x = (y->left == nil) ? y->right : y->left;

    if (root == (x->parent = y->parent))
        root->left = x;

    else
    {
        if (y == y->parent->left)

```

```

        y->parent->left = x;
    else
        y->parent->right = x;
}

if (y != z)
{
    if (!(y->color))
        rb_tree_del_fixup(tree, x);
    y->left = z->left;
    y->right = z->right;
    y->parent = z->parent;
    y->color = z->color;
    z->left->parent = z->right->parent = y;

    if (z->parent->left == z)
        z->parent->left = y;

    else
        z->parent->right = y;
    free(z);
}

else
{
    if (!(y->color))
        rb_tree_del_fixup(tree, x);
    free(y);
}
}

int main(void)
{
    int i, size;
    int data[21] = { 0 };

    rb_tree *rbt = NULL;
    rb_node *find = NULL;

    srand(time(NULL));
    size = sizeof(data) / sizeof(int) - 1;

    init_rand_arr(data, size);
    rbt = rb_tree_create();

    for (i = 0; i < size; i++)
        rb_tree_ins(&rbt, data[i]);
    rb_tree_print(rbt);
    find = rb_tree_find(rbt, data[5]);
    rb_tree_del(rbt, find);
    printf("\nAfter Delete\n");

    rb_tree_print(rbt);
    return 0;
}

```

--