

TI DSP, MCU 및 Xilinx Zynq FPGA 프로그램 전문가 과정

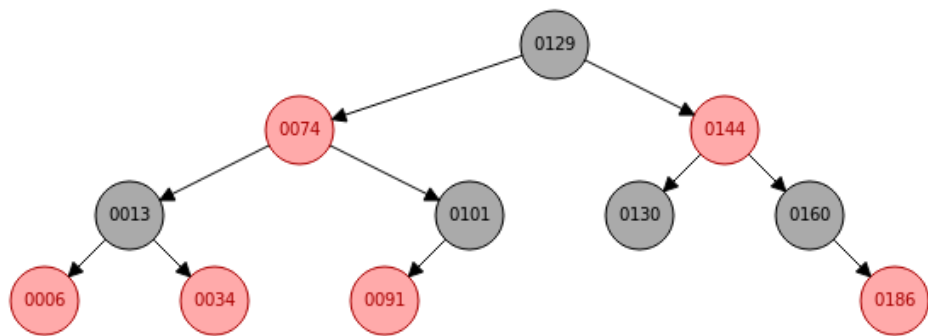
강사 - Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 - 은태영

zero_bird@naver.com

RB Tree_개념



❖ 시작점은 항상 검정이다.

❖ 잎사귀 노드는 어느 위치에 있던, 거치는 검정색의 개수가 서로 같다.

1. 빨강이 연속해서 두개가 위치하면 회전하거나 색을 변경한다.
2. 현재 기준점에서 부모 노드와 삼촌의 색상이 같으면 색상만 변경된다.
3. 할아버지가 빨강색이 되고, 그 자식들은 검정색이 된다.
4. 1번의 조건이 만족하는데 2번이 만족하지 않을 경우, 회전한다.

RB Tree - 구조체

```
tewill@tewill-B85M-D3H: ~/my_proj
#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;

1,1 Top
```

- ❖ node와 tree로 이루어진 구조체를 선언한다.
- ❖ node는 data와 color가 존재하며, 기존 이진 트리
의 왼쪽과 오른쪽을 나타내는 포인터에서 부모의
주소를 나타내는 포인터를 추가한 형태이다.
- ❖ color는 BLACK = 0 과 RED = 1 로 이루어진다.
- ❖ tree는 실제 데이터 주소를 보관하는 root 과 비어
있는 값 nil로 이루어져 있다.

RB Tree - main

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
}
```

445,0-1 Bot

- ❖ for 문 사용을 위한 i 와 data 의 사이즈를 알기 위해 size 를 int 로 선언한다.
- ❖ data 값을 넣기 위해 int [] 로 선언을 한다.
- ❖ RB 트리를 사용하기 위해, tree 와 node 를 선언한다.
- ❖ 난수를 생성하기 위해 srand 를 사용한다.
- ❖ data 의 사이즈를 sizeof 를 이용해 구한다.
- ❖ init_rand_arr 을 호출한다.

RB Tree - init_rand_arr / is_dup

```
tewill@tewill-B85M-D3H: ~/my_proj
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() % 15 + 1;
        arr[i] = rand() % 200 + 1;

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

25,1 5%

- ❖ 배열과 배열의 길이를 받아온다.
- ❖ for 문을 위해 i 를 선언한다.
- ❖ 배열의 길이만큼 반복하여, 각 배열 안에 랜덤한 값을 집어 넣는다.
- ❖ 이때 if 문과 is_dup 함수를 통하여 해당 배열에 중복되는 값이 있는지 확인하고, 있을 경우, goto 를 이용하여 해당 위치에 값을 다시 입력한다.

RB Tree - main

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
}
```

445,0-1 Bot

- ❖ init_rand_arr 이 종료된 후, rb_tree_create 함수를 호출한다.
- ❖ 호출 후, 반환값인 주소를 rbt에 저장한다.

RB Tree - rb_tree_create

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
}
```

217,1 43%

- ❖ 시작 위치를 잡아 줄, rbt 와 tmp 를 생성한다.
- ❖ malloc 을 이용하여, heap 으로 메모리를 할당 후, 해당 주소를 rbt 에 저장한다.
- ❖ tmp 와 rbt 가 가리키는 nil 에 malloc 을 통해 node 의 메모리를 할당 후, 그 주소를 저장한다.
- ❖ tmp 의 주소를 통하여 왼쪽, 오른쪽, 부모 모두 초기화한다.
- ❖ tmp 의 색을 기본적으로 검정으로 설정한다.
- ❖ data 를 0으로 초기화 한다.
- ❖ 위의 행위를 반복하여 root 역시 메모리 할당을 실시한다.
- ❖ 해당 주소를 모두 갖고 있는 rbt를 리턴한다.

RB Tree - main

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
}
```

445,0-1 Bot

- ❖ for 문을 이용하여 배열의 크기만큼 반복한다.
- ❖ 반복 내용으로는 rb_tree_ins 함수를 호출한다.

RB Tree - rb_tree_ins

```
tewill@tewill-B85M-D3H: ~/my_proj
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;

    127,1    27%
```

- ❖ node 에 대한 포인터를 3개 생성한다.
- ❖ node 를 메모리 할당 후, 이를 x 에 저장한다.
- ❖ x의 데이터에 인자로 받아 온 data를 집어넣는다.
- ❖ rb_tree_ins_helper 함수를 호출한다.

RB Tree - rb_tree_ins_helper

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
}
```

98,1 21%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
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

```

141,1-8 31%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
}
```

198,0-1 37%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
}
```

98,1 21%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
}
```

54,1 11%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
}
```

96,1 16%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
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);
}

219,1 48%
```


RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
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;

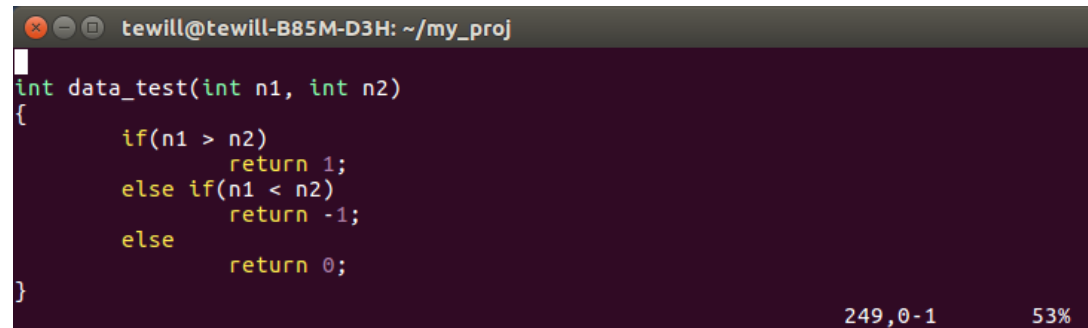
        if(x == nil)
            return 0;

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

    return x;
}
```

259,0-1 57%

RB Tree



```
tewill@tewill-B85M-D3H: ~/my_proj
int data_test(int n1, int n2)
{
    if(n1 > n2)
        return 1;
    else if(n1 < n2)
        return -1;
    else
        return 0;
}
```

249,0-1 53%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
    }
}
```

397,0-1 87%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
}
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);
}
}
```

419,1-8 93%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
    }
}
```

316,1 64%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
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;
            }
        }
    }
}
```

359,1-8 73%

RB Tree

```
tewill@tewill-B85M-D3H: ~/my_proj
}
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;
}
```

376,1-8 81%

