EDDI
Electronic Design
Development Institute

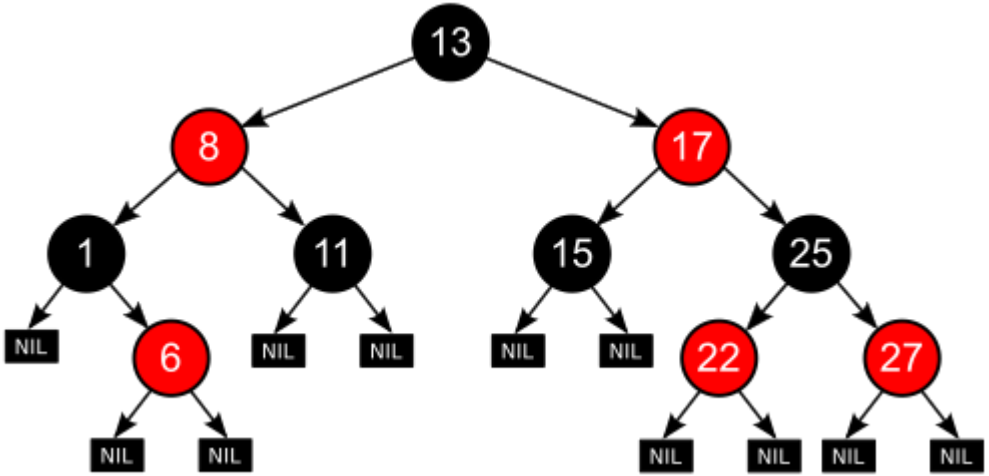
에디로봇아카데미
임베디드 마스터 Lv2 과정

제 1기

2021. 11. 19

김태훈

RED BLACK TREE



AVL Tree가 balance factor를 1 초과가 안되도록 관리하기 때문에 insert, delete때마다 수많은 연산이 발생하게 된다.

이런 문제를 방지하기 위해서
한쪽 tree보다 다른쪽 tree가 2배이상 못 크도록 설계한 tree가 red black tree이다.

현대에 많이 쓰이는 자료구조이다.

RED BLACK TREE

EDDI
Electronic Design Development Institute

Red-Black RULES:

1/ each node must be either **RED** or **BLACK**

2/ the root of the tree must ALWAYS be **BLACK**

3/ two **RED** nodes can never appear in a row within the tree; a **RED** node must always have a **BLACK** parent node, and **BLACK** child nodes

4/ every branch path from the root node in the tree to a NULL pointer passes through the exact same number of **BLACK** nodes (this is also an unsuccessful search path)

RED BLACK TREE

EDDI
Electronic Design Development Institute

```
typedef struct _redblack redblack;
struct _redblack
{
    int header;
    int data;
    struct _redblack *parent;
    struct _redblack *left;
    struct _redblack *right;
    int color;
    int black_level;
};
```

Redblack tree의 struct입니다.
Color은 int로 해두었는데, RED/BLACK 2가지 값만 가지므로 space 관점에서는 엄청난 낭비(int는 32bit)이지만, 나중에 바꾸면 되므로 일단 이렇게 구현해뒀습니다.

Parent node가 있어야 구현이 쉬울 것 같아서 만들어 두었습니다.
Black level은 처음 struct를 만들때 구현에 필요할 것 같아서 만들어 두었는데, 정작 구현해보니 큰 쓸모는 없고 TEST 판별할 때 유용하게 사용했습니다.

RED BLACK TREE

EDDI
Electronic Design Development Institute

규칙을 기반으로 Test code 작성을 먼저 해보았습니다.
Node의 Insertion 할 때의 조건을 인터넷에서 Pseudo code나 알고리즘을 찾아서 구현하는 것보다 추론해보는 것이 더 도움될 것 같아서 다음과 같이 진행했습니다.

```
void redblack_property_test(redblack **root)
{
    // 1. The color of root is black
    // NUMBER 1 condition will be checked in print function.

    // this function will be inserted in print function.
    // 2. There is no adjacent red node
    if((*root)->color == RED)
    {
        if((*root)->parent)
            assert((*root)->parent->color == BLACK);
        if((*root)->left)
            assert((*root)->left->color == BLACK);
        if((*root)->right)
            assert((*root)->right->color == BLACK);
    }
    // 3. Same black_level
    // compare left and right black level

    if((*root)->right && (*root)->left)
    {
#if 0
        printf("(*root)->data = %d\n",(*root)->data);
        printf("(*root)->right->data = %d\n",(*root)->right->data);
        printf("(*root)->left->data = %d\n",(*root)->left->data);
        printf("(*root)->black_level = %d\n",(*root)->black_level);
        printf("(*root)->right->black_level = %d\n",(*root)->right->black_level);
        printf("(*root)->left->black_level = %d\n",(*root)->left->black_level);
#endif
        assert((*root)->right->black_level == (*root)->left->black_level);
    }
}
```

Red node는 연달아 오지 못한다는 규칙이 있기 때문에, 본인이 RED라면 어떤 노드의 child와 parent가 모두 black이어야 하는 test를 하나 생성했습니다.
그리고 왼쪽 경로의 black node의 수와 오른쪽 경로의 black node의 수가 같아야 한다는 점도 있어서 이 부분도 test에 넣었습니다.

RED BLACK TREE

EDDI
Electronic Design Development Institute

```
        }
        else
        {
            printf("parent = NULL\t");
        }
        if(tmp->left)
        {
            if(tmp->left->data == -1)
                printf("left = NIL\t");
            else
                printf("left = %4d\t", tmp->left->data);
        }
        else
        {
            printf("left = NULL\t");
        }

        if(tmp->right)
        {
            if(tmp->right->data == -1)
                printf("right = NIL\t");
            else
                printf("right = %4d\t", tmp->right->data);
        }
        else
        {
            printf("right = NULL\t");
        }

        printf("black level = %4d\t", tmp->black_level);

        if(tmp->color == RED)
        {
            printf("color = RED\n");
        }
        else
        {
            printf("color = BLACK\n");
        }
        print_redblack(tmp->right);
        redblack_property_test(&tmp);
    }
}
```

Test 함수는 print의 마지막에 넣어서
node를 출력할 때마다 node의
property를 check하게 설계했습니다.

RED BLACK TREE

EDDI
Electronic Design
Development Institute

```
// property 1 : The color of root node is BLACK
assert((root ? root->color : RED) == BLACK);
```

레드블랙트리의 특성 중에 root의 색은 검은색이라는 특성도 있어서 main 함수에 이렇게 삽입해두었습니다.

RED BLACK TREE

EDDI
Electronic Design Development Institute

```
int LR_flag = 0;
if(!(*root))
{
    // make NIL singleton
        NIL = (redblack *)malloc(sizeof(redblack));
        NIL->header = REDBLACK_TREE;
        NIL->parent = NULL;
        NIL->left = NULL;
        NIL->right = NULL;
        NIL->color = BLACK;
        NIL->black_level = 1;
    NIL->data = -1;
    goto create_node;
}
else
{
    NIL = (*root)->parent;
}
while((*root)!=NIL)
{
    if((*root)->color == BLACK)
    {
        black_cnt++;
    }
    if((*root)->data > data)
    {
        tmp_grandparent = tmp_parent;
        tmp_parent = root;
        root = &(*root)->left;
    }
    else if((*root)->data < data)
    {
        tmp_grandparent = tmp_parent;
        tmp_parent = root;
        root = &(*root)->right;
    }
}
create_node:
// tmp == NULL when first insert node
*root = create_redblack_node();
(*root)->data = data;
(*root)->parent = tmp_parent ? *tmp_parent : NIL;
(*root)->left = NIL;
(*root)->right = NIL;
update_black_level(root);
(*root)->color = tmp_parent ? RED : BLACK;
```

현재 insert만 구현에 성공한 상태이고, delete는 아직 구현중입니다.

먼저 NIL node를 하나만 생성하고, 그 instance를 계속 쓰기 위해서

꼼수(?)를 썼는데

방법은 한번 만들어두고 root node의 parent를 NIL로 설정해두는 것입니다..

궁금한 점은 NULL을 NIL로 취급해도 되는데, 구현상 NIL로 해야하는 이유가 있는지 궁금하네요.

물론 Property 상 NIL이 중요한 역할을 하지만 구현할때는 NULL이나 NIL로 하나 큰 차이가 있나 싶은데 구현 다 하고 나서 NULL로 구현해봐야겠습니다.

RED BLACK TREE

EDDI
Electronic Design Development Institute

```
if((*root)->parent->color == RED && (*root)->color == RED) // no adjacent red violation
{

    printf("RED violation occur\n");

        // compare brother of parent
        if((*tmp_uncle)->color == BLACK)
        {
            if(LR_flag == LL || LR_flag == RR)
            {

                printf("case : LL or RR \n");

                (*tmp_parent)->color = BLACK;
                //after recoloring, same black level violation occurs.
                //rotate
                rotate_middle = (*tmp_parent);
                //rotate_mine = (*tmp_mine);
                rotate_bro = (*tmp_grandparent);

                rotate_middle->parent = rotate_bro->parent;
                rotate_bro->parent = rotate_middle;
                *tmp_parent = *tmp_bro;
                *tmp_bro = rotate_bro;
                *tmp_grandparent = rotate_middle;
                if((*tmp_parent)->data != -1)
                    (*tmp_parent)->parent = rotate_bro; // NIL의 parent가 생기는 문제
                rotate_bro->color = RED;
                update_black_level(&rotate_bro);
                update_black_level(&rotate_middle);
            }
            else if(LR_flag == LR)
```

제가 찾은 방법은
삽입한 Node의 색이 RED이고(삽입할 때는 항상 RED)
부모의 색이 RED이면
인접한 node가 RED면 안 된다는 규칙을 위반했으므로

색칠을 다시 하든, 회전을 하든 해야 합니다.

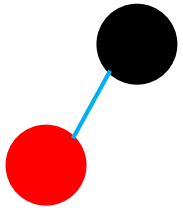
하나씩 예시를 확인해보면서 진행해보겠습니다.

RED BLACK TREE

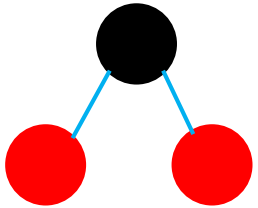
EDDI
Electronic Design
Development Institute

1. ROOT는 검은색

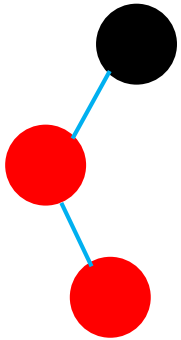
2. 다른 것 삽입



3-1. 반대쪽 child에 삽입할 경우



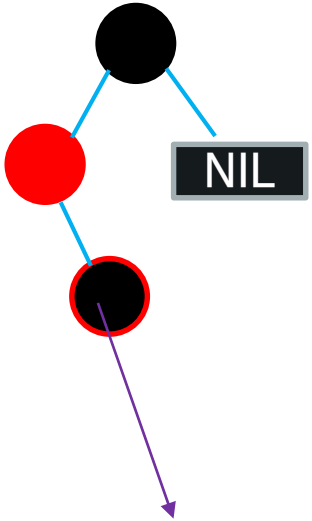
3-2. 이어서 삽입할 경우 : Violation



RED BLACK TREE

EDDI
Electronic Design Development Institute

3-2부터 보자.. 해결방법 : color 바꾸기?

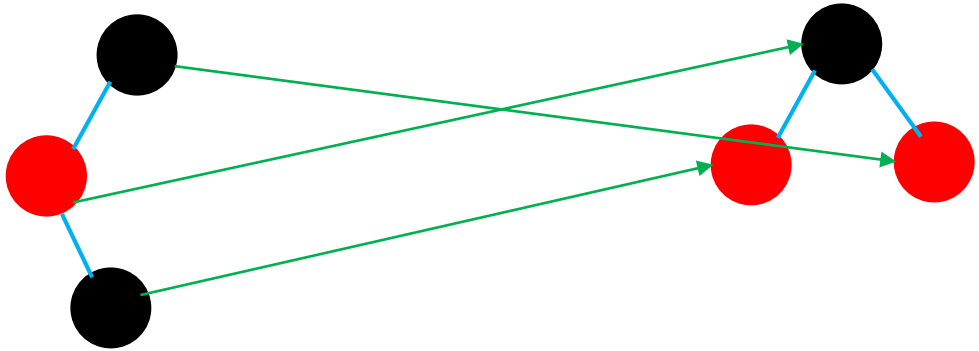


아래를 검은색으로 바꾸면 왼쪽경로로는 NIL까지 가는데 존재하는 black node가 하나 생기고, 오른쪽으로 가는 black node는 0개이므로 Black level violation이 된다. 따라서 위 케이스는 색을 바꿀 수 없으므로 회전해야한다.

RED BLACK TREE

EDDI
Electronic Design
Development Institute

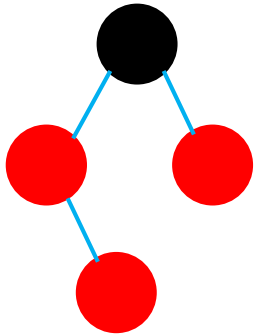
3-2 해결방법 : 회전!



RED BLACK TREE

EDDI
Electronic Design Development Institute

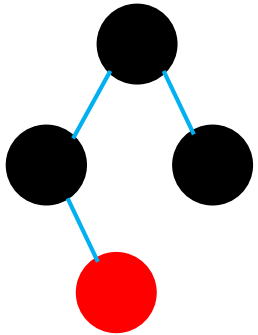
4-1이어서 삽입 : Violation



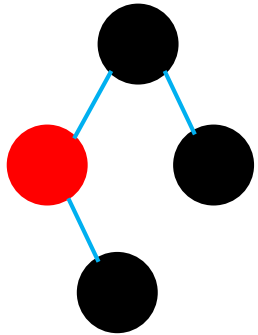
이것도 색을 바꾸면 되나?
색 바꾸는 게 리소스가 적게 들기 때문에 항상 먼저 고려해야 한다.

RED BLACK TREE

EDDI
Electronic Design Development Institute

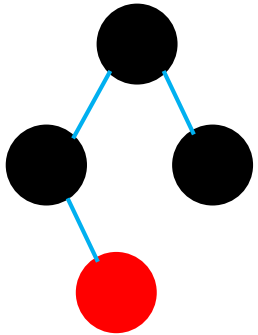


1번. Child를 다 검은색으로 바꾸기

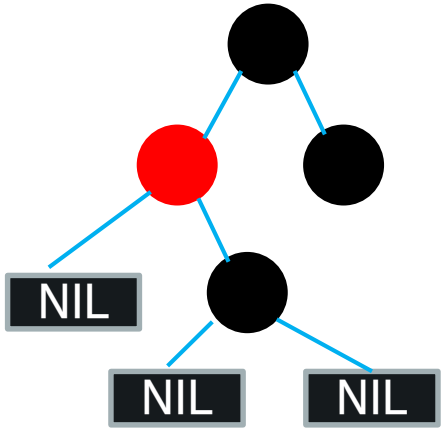


2번. 삽입한 node와 삼촌을 검은색으로 바꾸기

RED BLACK TREE

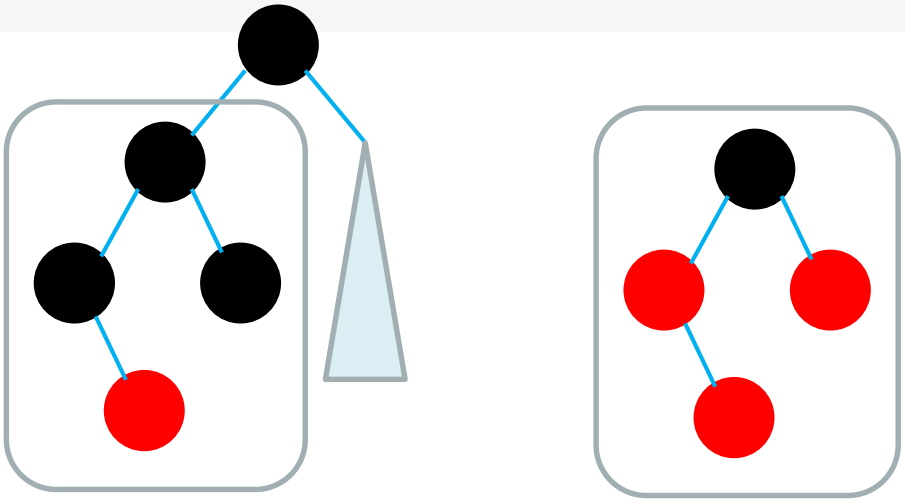


1번. Child를 다 검은색으로 바꾸기



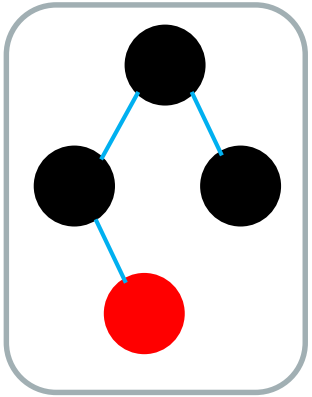
2번 되는 case 같지만 자세히 살펴보면 왼쪽 NIL로 가는 경로에는 검은 노드가 없지만 왼쪽 ->오른쪽 경로에는 검은노드가 하나 있어서 규칙 위반이다.

RED BLACK TREE

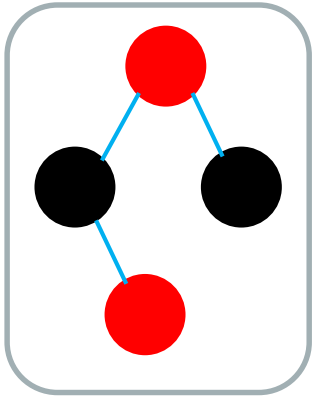


1번. Child를 다 검은색으로 바꾸기
-> 근데 이건.. 너무 다 검은색으로 바꿔서 RED가 많이 없어진다. 게다가 상윗단에 node가 있다고 가정해보자.
원래 black node가 1개 있었는데 왼쪽처럼 하면 한 층이 더 생긴다.
그럼 당연히 오른쪽 삼각형 node에서의 black level과 달라지므로 위반이다..

RED BLACK TREE



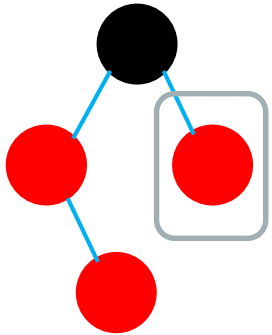
Parent가 ROOT일 때

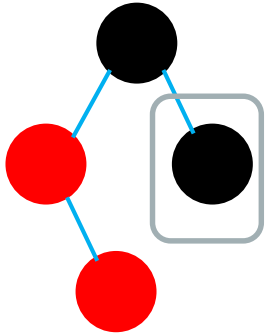


Parent가 ROOT가 아닐 때

이런 이유로, parent가 root인 경우만 제외하곤 parent를 red로 만들어야 한다.

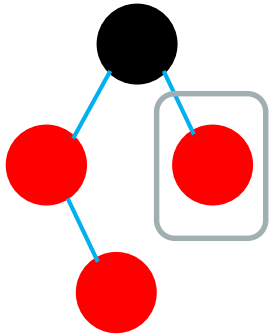
RED BLACK TREE

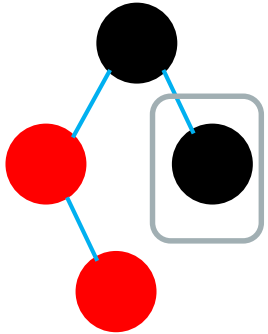




여태까지 여러 케이스를 봤는데, 잘 보면 공통점을 추려낼 수 있다.
삽입한 node의 uncle(삼촌) node의 색을 보면 회전인지, 색칠하기 인지 구분할 수 있다.

RED BLACK TREE





삼촌이 빨간색인 경우 : 색깔 바꾸기로도 충분히 해결됨.
삼촌이 검은색인 경우 : 색깔 바꾸면 안 되는 게, 왼쪽은 빨간색이 2개이므로 무조건 색 하나는 바뀌어야 하는데 그럼 왼쪽 black level이 1 올라간다. 기존에 black level이 양쪽 똑같았는데 왼쪽이 올라가면 맞춰주기 위해서 오른쪽도 올라가야 하는데, 오른쪽은 이미 검은색이라 올라갈 수가 없다. 그래서 회전해야 한다..

RED BLACK TREE

EDDI
Electronic Design Development Institute

```
if((*root)->parent->color == RED && (*root)->color == RED) // no adjacent red violation
{

    printf("RED violation occur\n");

        // compare brother of parent
        if((*tmp_uncle)->color == BLACK)
        {
            if(LR_flag == LL || LR_flag == RR)
            {

                printf("case : LL or RR \n");

                (*tmp_parent)->color = BLACK;
                //after recoloring, same black level violation occurs.
                //rotate
                rotate_middle = (*tmp_parent);
                //rotate_mine = (*tmp_mine);
                rotate_bro = (*tmp_grandparent);

                rotate_middle->parent = rotate_bro->parent;
                rotate_bro->parent = rotate_middle;
                *tmp_parent = *tmp_bro;
                *tmp_bro = rotate_bro;
                *tmp_grandparent = rotate_middle;
                if((*tmp_parent)->data != -1)
                    (*tmp_parent)->parent = rotate_bro; // NIL의 parent가 생기는 문제
                rotate_bro->color = RED;
                update_black_level(&rotate_bro);
                update_black_level(&rotate_middle);
            }
            else if(LR_flag == LR)
```

```
        }
        else // UNCLE is RED
        {

            printf("recoloring uncle and parent BLACK\n");

            if((*tmp_grandparent)!=(*root_backup))
                (*tmp_grandparent)->color = RED;
            (*tmp_parent)->color = BLACK;
            (*tmp_uncle)->color = BLACK;
            update_black_level(tmp_mine);
            update_black_level(tmp_uncle);
        }
```

구현상의 몇몇 이슈들을 해결하면 잘 된다.

RED BLACK TREE

EDDI
Electronic Design Development Institute

```
data = 9973     parent = 9974   left = 9972     right = NIL     black level =    2       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9974     parent = 9980   left = 9973     right = 9976    black level =    3       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9975     parent = 9976   left = NIL      right = NIL     black level =    2       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9976     parent = 9974   left = 9975     right = 9978    black level =    2       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9977     parent = 9978   left = NIL      right = NIL     black level =    1       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9978     parent = 9976   left = 9977     right = 9979    black level =    2       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9979     parent = 9978   left = NIL      right = NIL     black level =    1       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9980     parent = 9985   left = 9974     right = 9982    black level =    3       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9981     parent = 9982   left = NIL      right = NIL     black level =    2       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9982     parent = 9980   left = 9981     right = 9984    black level =    3       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9983     parent = 9984   left = NIL      right = NIL     black level =    1       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9984     parent = 9982   left = 9983     right = NIL     black level =    2       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9985     parent = 9971   left = 9980     right = 9991    black level =    4       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9986     parent = 9988   left = NIL      right = 9987    black level =    2       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9987     parent = 9986   left = NIL      right = NIL     black level =    1       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9988     parent = 9991   left = 9986     right = 9989    black level =    3       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9989     parent = 9988   left = NIL      right = 9990    black level =    2       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9990     parent = 9989   left = NIL      right = NIL     black level =    1       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9991     parent = 9985   left = 9988     right = 9997    black level =    3       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9992     parent = 9994   left = NIL      right = 9993    black level =    2       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9993     parent = 9992   left = NIL      right = NIL     black level =    1       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9994     parent = 9997   left = 9992     right = 9995    black level =    2       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9995     parent = 9994   left = NIL      right = 9996    black level =    2       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9996     parent = 9995   left = NIL      right = NIL     black level =    1       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9997     parent = 9991   left = 9994     right = 9999    black level =    3       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9998     parent = 9999   left = NIL      right = NIL     black level =    1       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 9999     parent = 9997   left = 9998     right = 10000   black level =    2       color = BLACK
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
data = 10000    parent = 9999   left = NIL      right = NIL     black level =    1       color = RED
data = NIL      parent = NULL   left = NULL     right = NULL    black level =    1       color = BLACK
```

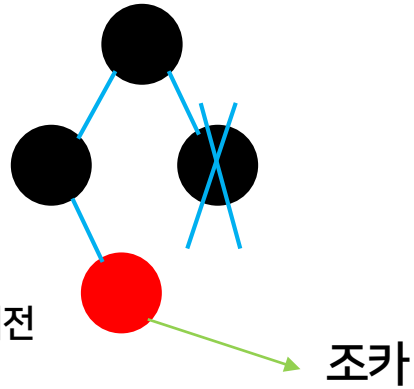
삽입의 경우 10000개 까지 규칙 위반 없이 잘 작동한다.

RED BLACK TREE

EDDI
Electronic Design
Development Institute

삭제?

현재까지 알아낸 것은,
1. RED 삭제하고 NIL이 오면: PASS
2. RED 삭제 RED가 오면 : RED NODE 조건에 의해서 불가능한 경우
3. RED 삭제 BLACK이 오면: 조카가 없으면 형제하고 부모하고 색 바꿈 조카 있으면 회전
4. BLACK 삭제 RED이 오면 : 조카가 없으면 형제 부모 색 바꿈 조카 있으면 회전
5. BLACK 삭제 NIL come : 조카가 없으면 형제 부모 색 바꿈 조카 있으면 회전
6. BLACK 삭제 BLACK come : 조카가 없으면 형제 부모 색 바꿈 조카 있으면 회전



삭제되는 node와 그 자리에 오는 node의 색을 보고 판별해야 한다.

시간관계상 더 알아내지 못했습니다.