**Homework 3B report**

**2014313303 홍태하(개발환경 : Linux)**

1. **알고리즘**

먼저 rbt\_create함수를 통해 트리 T를 생성한다. 그 후 인풋으로 들어오는 값들을 key값으로 가지는 노드를 생성하여 트리 T에 insert한다. insert는 기존의 binary tree와 같이 먼저 새로 insert하는 노드와 기존 트리의 노드의 key값을 비교하여 새로운 노드의 위치를 찾는다. 하지만 redblacktree는 여기에서 끝나지 않고 insert\_rbt\_fixup이라는 함수를 통해 트리를 재구성한다. 재구성함으로써 binary tree에서 한쪽으로만 노드가 생기는 상황을 만들지 않는다.

insert\_rbt\_fixup은 먼저 새로운 노드를 red인 상태로 insert하면서 시작한다. 새로운 노드를 z라고 하면, z의 부모가 black인 경우 아무 문제가 없으므로 그대로 끝난다. 하지만 z의 부모가 red이면 red가 연속해서 나오기 때문에 문제가 된다. 이 때 z의 부모가 z의 조부모의 왼쪽 자식인지 오른쪽 자식인지 각각 3가지 케이스가 있다.

오른쪽 자식일 경우 왼쪽 자식일 경우에서 left, right만 바꿔주면 되므로 왼쪽 자식일 경우만 설명하겠다. 3가지 케이스가 있는데 먼저 z부모의 형제노드도 red인 경우이다. 이 경우에는 z의 부모와 z의 삼촌(z부모의 형제)를 모두 black으로 바꿔주고 z를 z의 조부모로 바꾼 후 다시 fixup을 시작한다. 두 번째, 세 번째 경우는 z의 삼촌노드가 black인 경우이다. null노드도 black이므로 z의 삼촌노드가 없을 경우(null)도 이 케이스에 해당한다. 두 번째 경우는 그 중에서도 z가 z부모의 오른쪽 자식일 경우이다. 이 경우에는 z의 부모에서 left rotate를 해준다. left rotate는 x에서 left rotate를 해줬을 때 x의 오른쪽 자식 노드를 y라고 하면 y의 왼쪽 자식을 x의 오른쪽 자식으로 붙여주고 x를 y의 왼쪽 자식으로 바꿔주는 것이다. right rotate는 이것에서 왼쪽, 오른쪽을 바꿔주면 된다. 세 번째 경우는 z가 z부모의 왼쪽 자식일 경우이다. 두 번째 경우를 거치고 나면 자동으로 세 번째 경우의 상태가 된다. 이 경우에는 z의 부모를 black으로 바꾸고 z의 조부모를 red로 바꿔준 후 z의 조부모에서 right rotate를 한다. 세 번째 과정을 거치면 첫 번째 과정과는 다르게 다시 fixup을 하지 않아도 된다. 마지막으로 루트 노드를 black으로 바꿔주면 모든 과정이 끝난다.

위와 같은 방법으로 트리를 생성하고 insert를 통해 노드를 생성해서 insert\_rbt와 insert\_rbt\_fixup을 반복하면 redblacktree가 완성된다. redblacktree를 완성한 후 black은 <>, red는 --사이에 key값을 넣어서 preorder과 inorder로 출력한다.

1. **코드 설명.**
2. **rotate\_left 함수**

void rotate\_left(rbt\_tree \*T, rbt\_node \*x)

T라는 트리 안에 있는 노드 x에서 x의 오른쪽 자식 노드를 y라고 하면 y의 왼쪽 자식을 x의 오른쪽 자식으로 붙여주고 x를 y의 왼쪽 자식으로 바꿔주는 것이다.

1. **rotate\_right 함수**

void rotate\_right(rbt\_tree \*T, rbt\_node \*x)

T라는 트리 안에 있는 노드 x에서 x의 왼쪽 자식 노드를 y라고 하면 y의 오른쪽 자식을 x의 왼쪽 자식으로 붙여주고 x를 y의 오른쪽 자식으로 바꿔주는 것이다.

1. **rbt\_create 함수**

rbt\_tree\* rbt\_create()

rbt\_tree를 생성하고 초기화 해준다. 구조체 rbt\_tree에는 root만 있으므로 root에 null값을 넣어주고 생성된 트리의 주소값을 리턴한다.

1. **insert 함수**

rbt\_node\* insert(rbt\_tree \*T, int key)

rbt\_node를 생성하고 key값을 넣어 초기화 해준다. 그 후 생성된 노드를 insert\_rbt함수를 통해서 트리 T에 insert한다. 마지막으로 노드의 주소값을 리턴한다.

1. **insert\_rbt 함수**

void insert\_rbt(rbt\_tree \*T, rbt\_node \*z)

새로 insert하는 노드 z의 위치를 찾아 트리 T에 넣어준 후 insert\_rbt\_fixup 함수를 통해 redblacktree를 재구성해준다. 노드의 위치를 찾는 것은 binary search의 insert와 같다. 처음 insert 된 노드는 모두 red로 시작해서 fixup된다.

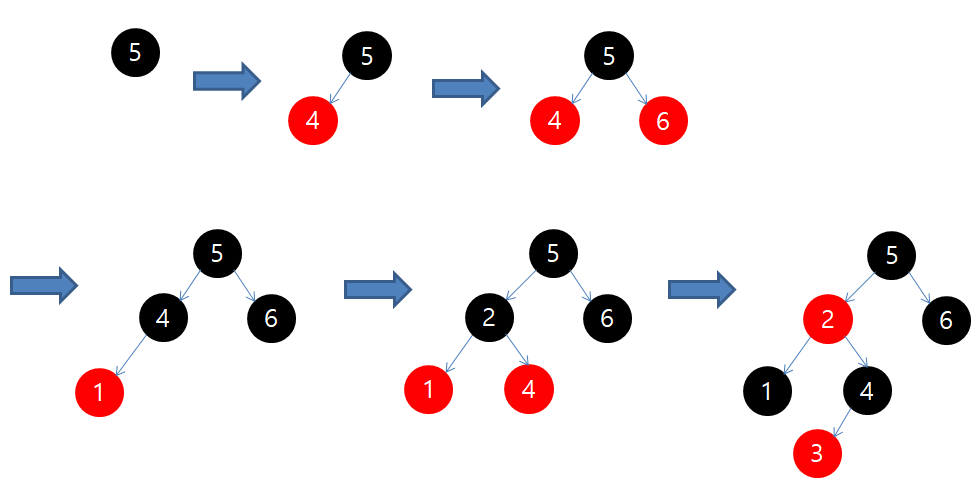
1. **insert\_rbt\_fixup 함수**

void insert\_rbt\_fixup(rbt\_tree \*T, rbt\_node \*z)

트리 T에 노드 z를 insert하고 redblacktree의 조건에 맞도록 fixup해주는 함수이다. 총 6가지 케이스로 나뉘어서 진행된다. 위의 알고리즘 부분에서 설명한 것과 같이 redblacktree를 완성한다.

1. **예시**

**6**

**5 4 6 1 2 3** 의 인풋이 들어왔을 경우 5 4 6 1 2 3의 순서로 redblacktree가 insert되어 만들어진다. 각각의 노드가 추가될 때를 보면 다음과 같다.

따라서 결과값은 다음과 같다.

<5>-2-<1><4>-3-<6>

<1>-2—3-<4><5><6>