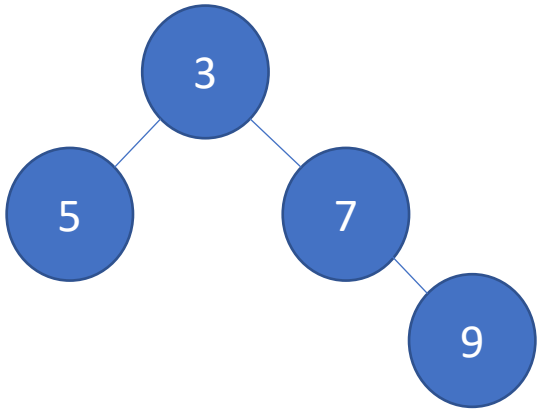
Red-Black Tree

J075 박상신

목차

Red-Black Tree 란

Binary Search Tree란

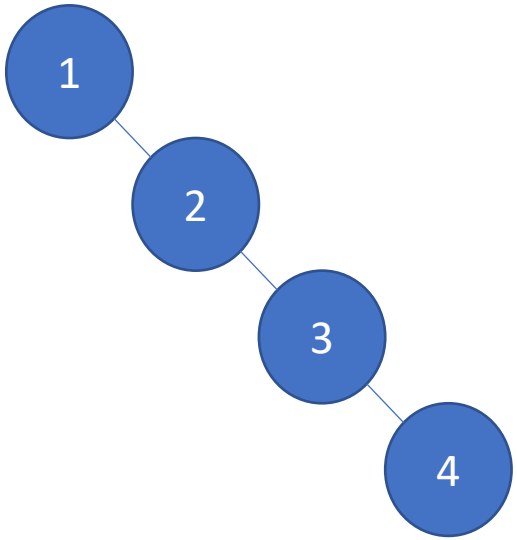


	ArrayList	LinkedList
get()	O(1)	O(n)
add()	O(1)	O(1) amortized
remove()	O(n)	O(n)

Binary : 자식 노드가 2개
일정한 규칙을 가지고 자료를 저장해서 search, insert, delete시 배열, list에 비해 장점이 있음.

Red-Black Tree 란

Binary Search Tree의 단점



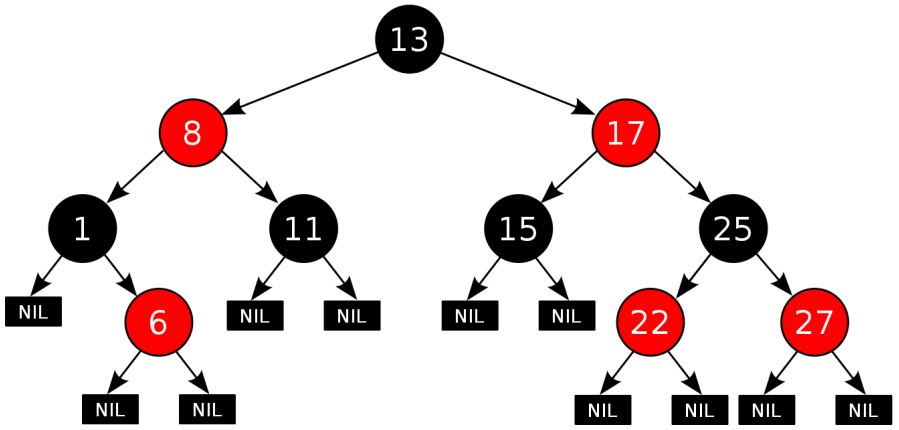
트리의 균형을 잡아 O(logN)의 이점을
활용할 수 있도록 할 필요성

Red-Black Tree 란

Red-Black Tree

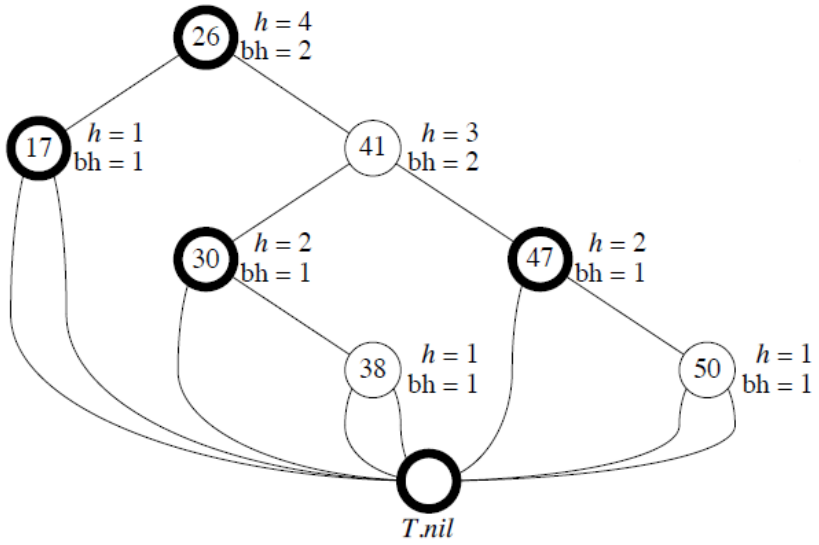
- Binary Search Tree의 일종 – 삽입, 삭제, 검색 O(logN)
- 균형 잡힌 트리 : 높이가 logN 이 되도록 유지
- 실 사용에서 효율적, 최악의 경우에도 우수한 실행 시간. 연관 배열(map) 구현하는데 쓰임.

```
struct RedBlackNode{
   int key;
   enum { red, black } color;
   RedBlackNOde *left, *right, *parent;
};
```



Red-Black Tree Rule

1. 모든 노드는 RED이거나 BLACK이다.
2. Root노드는 BLACK이다.
3. 모든 leap노드는 BLACK이다.
4. RED의 자식은 모두 BLACK이다. (red – red 금지)
5. 각 노드로부터 그 노드의 자손인 리프로 가는 경로들은 모두 같은 수의 BLACK노드를 포함한다.



4번조건 : insert
5번조건 : delete

Red-Black Tree – Left(Right) Rotation

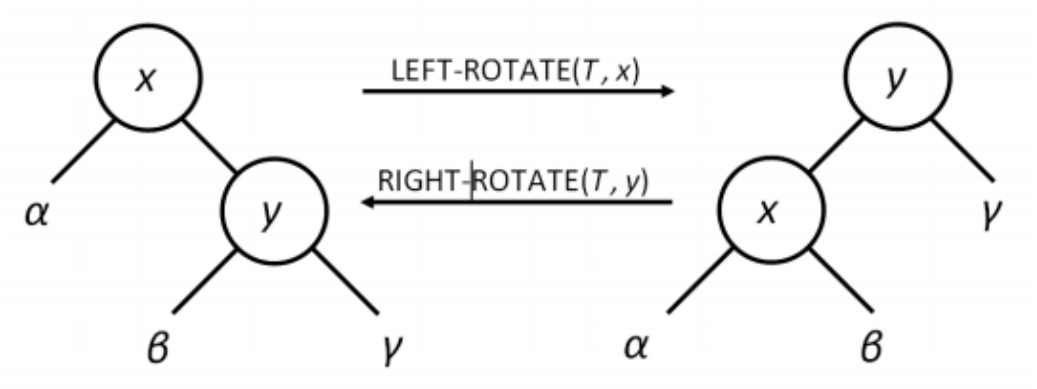
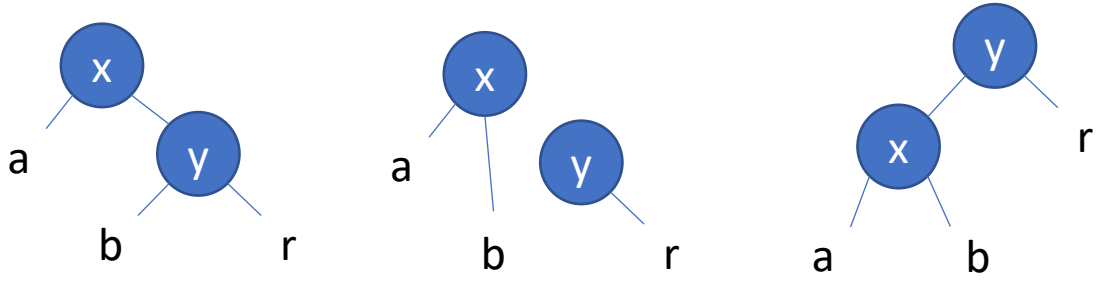


Figure 2. Rotations in a Red-Black tree



Red-Black Tree – Insert

```
RB-INSERT(T, z)
    y ← null
    x ← T->root
    while x ≠ null
            do y ← x
                    if z->key < x->key
                            then x ← x->left
                            else x ← x->right
    z->p ← y
    if y = null
            then T->root ← z
             else if z->key < y->key
                     then y->left ← z
                     else y->right ← z
    z->left ← null
    z->right ← null
    z->color ← RED
    RB-INSERT-FIXUP(T, z)
```

Binary Search Tree
Insert 알고리즘과 동일

삽입하는 node를 RED로 지정

Red-Black Tree – Insert

RB-INSERT-FIXUP 설계

```
z->color ← RED
RB-INSERT-FIXUP(T, z)
```

삽입하는 node를 RED로 지정

1. 모든 노드는 RED이거나 BLACK이다.
2. Root노드는 BLACK이다.
   -> z가 root노드라면 위반 -> red를 삽입 후 black으로 바꿔주면 간단 해결
3. 모든 leap노드는 BLACK이다.
4. RED의 자식은 모두 BLACK이다. (red – red 금지)
   -> z의 부모가 red이면 위반
5. 각 노드로부터 그 노드의 자손인 리프로 가는 경로들은 모두 같은 수의 BLACK노드를 포함한다.

Red-Black Tree – Insert

RB-INSERT-FIXUP의 종료 조건

- 부모노드 p[z]가 black이 되면 종료.
- 조건 2 위반일 경우 z를 black으로 바꿔주고 종료.

조건 2는 간단하니 조건 4 해결을 알아보자.

```
RB-INSERT-FIXUP(T, z)
while  z->p->color  = RED
        do if z->p =  z->p->p->left
                then y ← z->p->p->right
                     if y->color = RED
                          then    z->p->color ← BLACK                    Case 1
                                  y->color ← BLACK                       Case 1
                                  z->p->p->color ← RED                   Case 1
                                  z ← z->p->p                            Case 1
                          else if z =  z->p->right
                                  then    z ← z->p                       Case 2
                                          LEFT-ROTATE(T, z)              Case 2
                                          z->p->color ← BLACK            Case 3
                                          z->p->p->color ← RED           Case 3
                                          RIGHT-ROTATE(T, z->p->p)       Case 3
                else (same as then clause with "right" and "left" exchanged)
T->root->color ← BLACK
```

Red-Black Tree – Insert

Red-red violation 6가지 case

- 1, 2, 3 케이스 : p[z]가 p[p[z]]의 왼쪽 자식 (p[p[z]]는 반드시 존재)
- 4, 5, 6 케이스는 각각 1,2,3케이스의 대칭

Case 1. 삼촌 노드가 RED일 경우

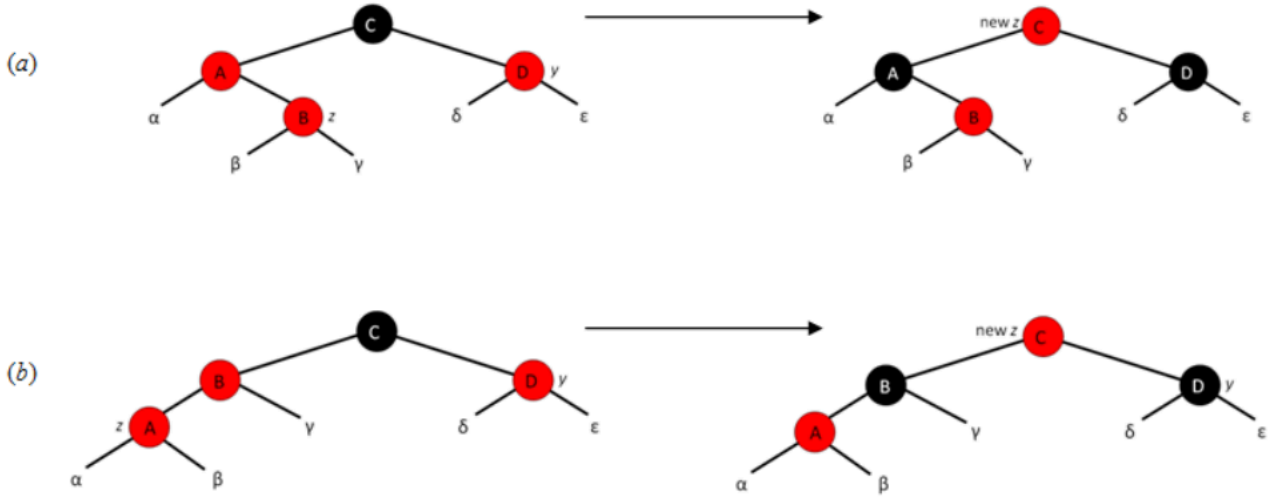


Figure 4. Cases 1 of rotations in a Red-Black tree

z의 아버지, 삼촌의 color(red)와 할아버지(black) 의 color를 switch하고 할아버지가 z가 된다.
새로운 z가 root면 black으로 바꿔줘서 쉽게 해결
새로운 z의 부모가 red라면 다시 red-red violation을 해결

Red-Black Tree – Insert

Case 2, 3. 삼촌 노드가 BLACK일 경우

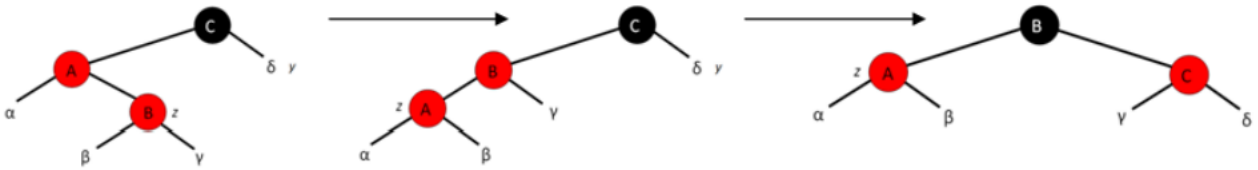


Figure 5. Cases 2 and 3 of rotations in a Red-Black tree

Case 2. z가 부모노드의 오른쪽 자식일 경우
-> p[z]에 대해 left-rotation한 후 p[z]를 z로 변경
-> case 3가 된다.

Case 3. z가 부모노드의 왼쪽 자식일 경우
-> p[z]를 black, p[p[z]]를 red로 변경
-> p[p[z]] 에 대하여 right-rotation

Red-Black Tree – Delete

```
RB-DELETE(T, z)
 if z->left = null or z->right = null
        then y ← z
        else y ← TREE-SUCCESSOR(z)
if y->left ≠ null
        then x ← y->left
        else x ← y->right
 x->p ← y->p
if y->p = null
        then T->root ← x
        else if y = y->p->left
                then y->p->left ← x
                else y->p->right ← x
if y 3≠ z
         then z->key ← y->key
                copy y's satellite data into z
if y->color = BLACK
        then RB-DELETE-FIXUP(T, x)
return y
```

Binary Search Tree
Delete 알고리즘과 동일

삭제한 node가 BLACK일경우
RB-DELETE-FIXUP 실행

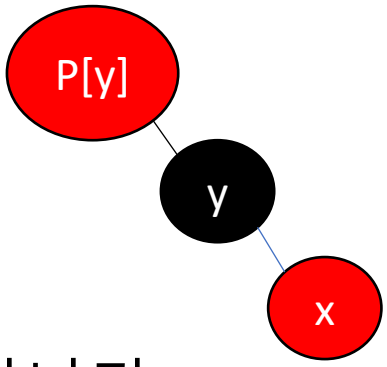
Red-Black Tree – Delete

RB-Delete-FIXUP 설계

```
if y->color = BLACK
        then RB-DELETE-FIXUP(T, x)
```

X는 nil or red or black
Red인 경우 쉽게 해결

1. 모든 노드는 RED이거나 BLACK이다.
2. Y가 root이고 x가 red인 경우 위반
   -> root가 red이면 black으로 바꿔주면 간단 해결
3. 모든 leap노드는 BLACK이다.
4. Y의 부모가 red일 경우 위반
   -> x를 black으로 바꿔주면 해결 가능.
5. 원래 y를 포함했던 모든 경로 -> black하나가 부족해서 black height에 문제가 생김.
   -> 가장 큰 문제

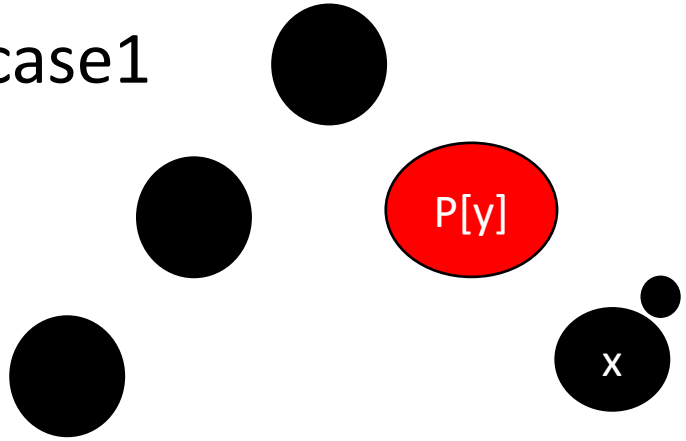


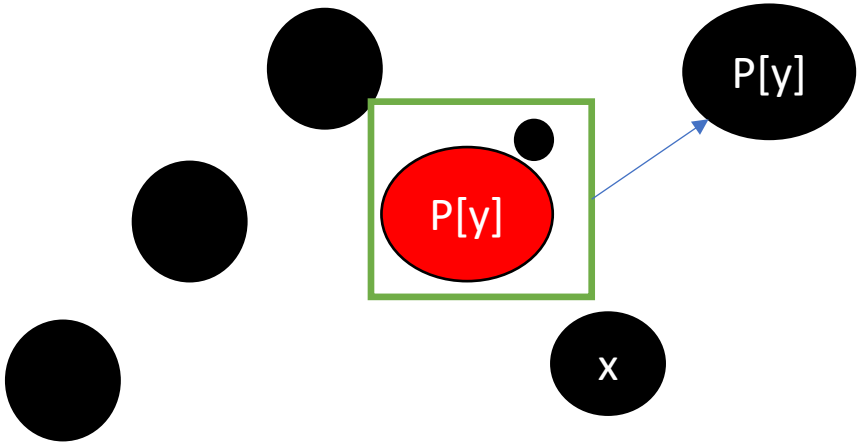
Red-Black Tree – Delete

5번 문제 해결 아이디어

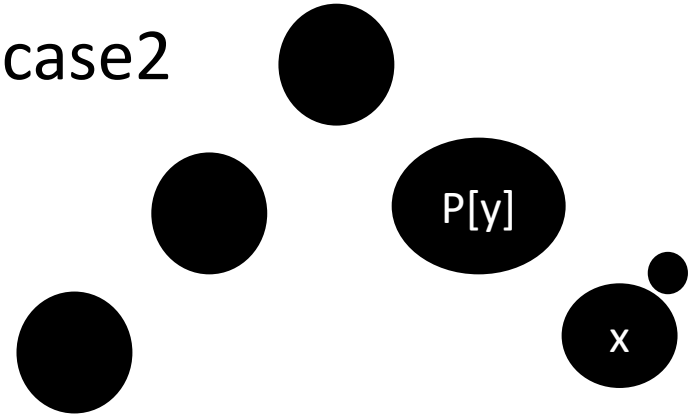
Extra black을 준다.

case1





case2



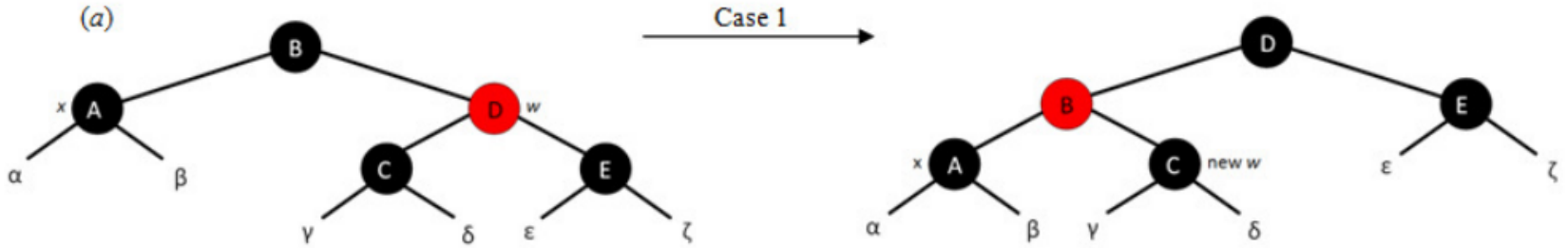
Extra black을 트리의 위쪽으로 올려보냄
1. Red를 만나면 black으로 바꾸고 종료
2. X가 root가 되면 extra black 제거

Red-Black Tree – Delete

8가지 case

- 1, 2, 3, 4 케이스 : x가 부모의 왼쪽 자식일 경우.
- 5~8 케이스는 1~4케이스의 대칭

Case 1. 형제노드가 red인 경우



X는 현재 double black.

형제노드가 red이기 때문에 그 자식은 black이다.
W를 black으로, p[x]를 red로 변경
p[x]에 대해 left-rotation 적용.

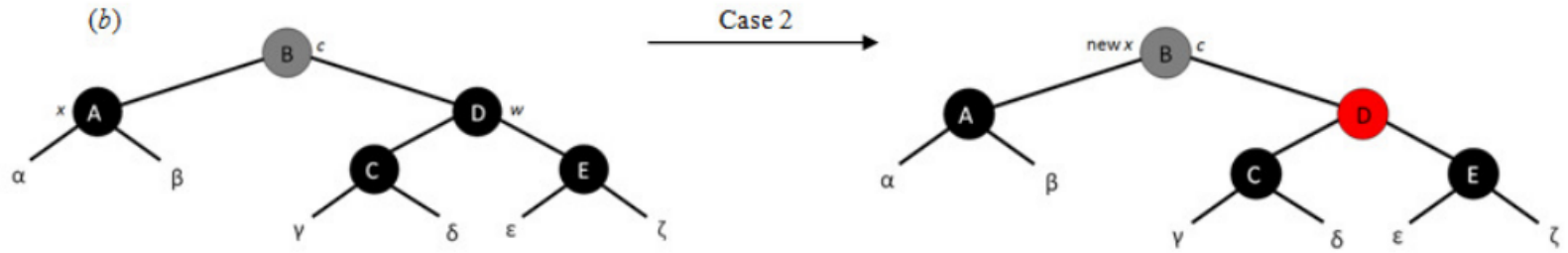
X는 본래 height보다 한칸 아래로 내려온다. (insert의 경우 위로 올려보내려 했던 것 과는 반대.)
2보 전진을 위한 1보 후퇴. Case 2, 3, 4 에서 해결.

Red-Black Tree – Delete

Case 2, 3, 4 형제노드가 black인데
- case 2 : 자식 둘 다 black인 경우
- case 3 : 왼쪽 자식이 red인 경우
- case 4 : 오른쪽 자식이 red인 경우

Red-Black Tree – Delete

Case 2. 형제노드가 black, 형제노드의 자식도 black인 경우



P[x]는 회색 -> Unknown color

P[x]가 x의 extra black과 w의 black을 가져감. (자기 자식 black하나씩 가져갔기 때문에 아래쪽은 문제 없음.)

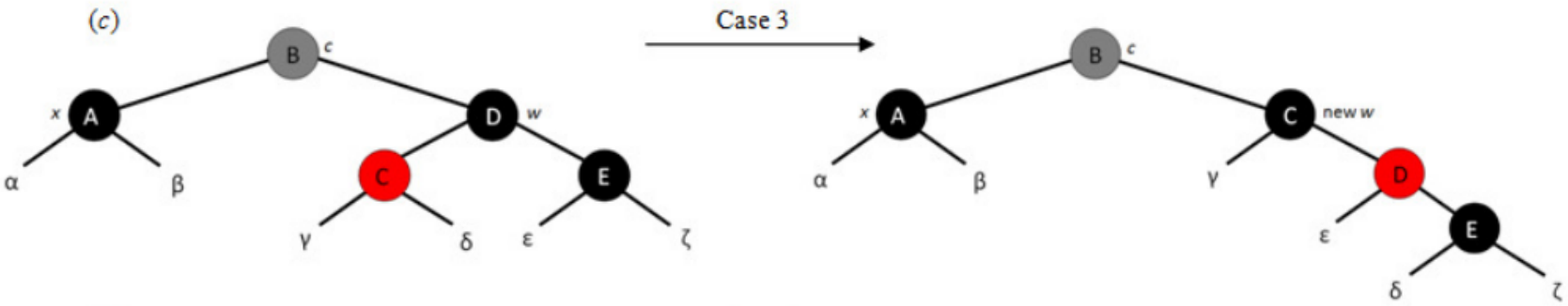
P[x]는 새로운 x가 된다

만약 p[x]가 red였다면 extrablack을 갖고 black으로 색 변경후 종료. (case 1에서 넘어온 case2라면 반드시 이 경우)

P[x]가 black이라면 extrablack을 갖고 있는 새로운 black x가 된다.

Red-Black Tree – Delete

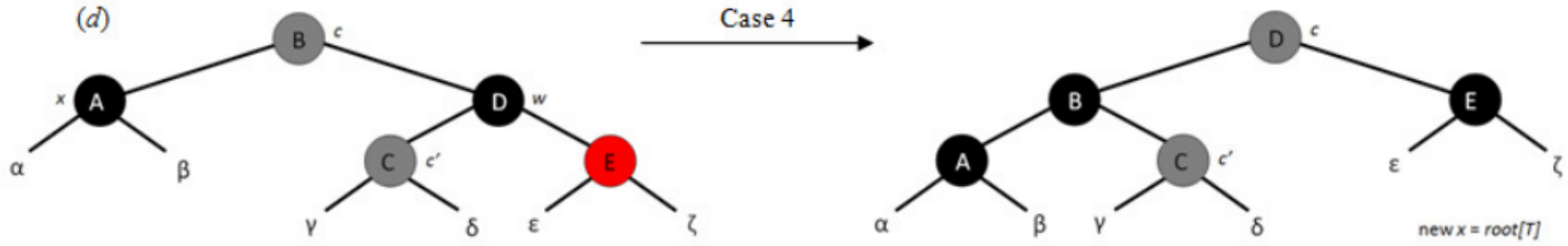
Case 3. 형제노드가 black, 형제노드의 왼쪽 자식이 red



W를 red로 바꾸고 w의 왼쪽 자식을 black으로 바꾼다.
W에 대해 right-rotation 적용한다.
Case 4로 넘어간다.

Red-Black Tree – Delete

Case 4. 형제노드가 black, 형제노드의 오른쪽 자식이 red



C와 c'는 unknown color

W의 색을 현재 p[x]의 색으로 변경
P[x]를 black으로, w의 오른자식을 black으로
P[x]에 대해서 left-rotation 적용
X의 extra-black을 제거하고 종료

참고자료

Redblacktree 권오흠 교수님 강의
http://software.ucv.ro/~mburicea/lab8ASD.pdf

Red black tree 책
http://software.ucv.ro/~mburicea/lab8ASD.pdf