1 Red black trees

Definition Tree is red black if:

1. Every node is red or black.
2. The root is black.
3. Every leaf is black.
4. If a node is red, then both its children are black.
5. For each node, all paths from the node to descendant leaves contain the same number of black nodes.

The rule 2 is sometimes is omitted, because the root can always be changed from red to black.

Red node can have either zero or two children.

Child subtree of any node x is at most twice longer than subtree of it's sibling (follows from properties 4 and 5). Subtree in any node x has at least $2^{h_b(x)} - 1$ internal nodes (proof in [1]). Red black tree with n internal nodes has height at most $2\lg(n+1)$.

If RB tree T has n nodes, then maximum number of red nodes is $2n/3$ (leaves not calculated). Maximal number is reached when nodes on levels $2, 4, \ldots, h(T)$, $h(T)$ is odd, are colored red and tree T is full (each node has both children).

It is considered that all leafs have sentinels as children (black nodes with leafs as parents and no children). Thus, algorithm for deleting will be easier to implement. Notation for a sentinel of a tree T is $s(T)$.

1.1 Inserting

Newly inserted node z is colored red. If (after being inserted) its parent is black, then the process is over. Otherwise, properties 2 and 4 can be violated (for details see [1]), so recolorings and rotations have to be made. There are three cases when red black properties are violated and another three symmetric to those ones. Let be y the z's uncle i.e. node such that $y = r(p(p(z)))$.

L1 y is red: then $p(p(z))$ is black, $p(z)$ is red $\Rightarrow y, p(z)$ can be colored black and $p(p(z))$ red. The process continues on $p(p(z))$.
L2 y is black and $z = r(p(z)) \Rightarrow$ left rotation of w transforms it to L3 case (where w gets role of z).
L3 y is black and $z = l(p(z)) \Rightarrow$ color $p(z)$ into black, $p(p(z))$ into red and rotate right $p(p(z))$; thus, tree has the correct red-black properties and process finishes.

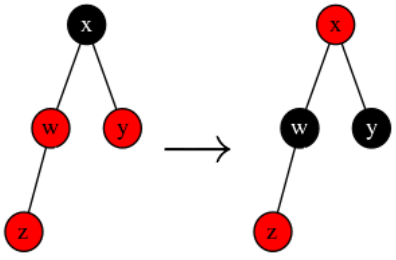

Figure 1: L1 case

Symmetric cases are for $y = l(p(p(z)))$:

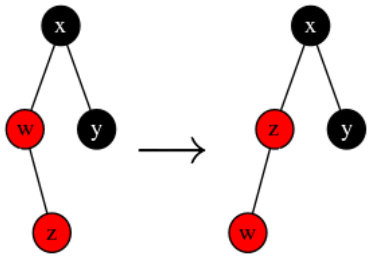

Figure 2: L2 case transformed to L3 (where w gets role of z)

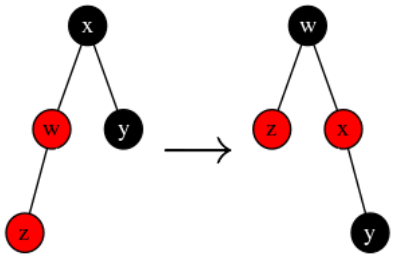

Figure 3: L3 case which fixes the red-black properties

R1 y is red: then $p(p(z))$ is black, $p(z)$ is red $\Rightarrow y, p(z)$ can be colored black and $p(p(z))$ red. The process is continued on $p(p(z))$.
R2 y is black and $z = l(p(z)) \Rightarrow$ right rotation of w transforms this case to case R3.
R3 y is black and $z = r(p(z)) \Rightarrow$ color $p(z)$ into black, $p(p(z))$ into red and rotate left $p(p(z))$.

Complexity: $O(\lg n)$ **Input:** key K to insert into tree T
Output: none

```
insert(K)
  if r_t(T) = null
    new z
    k(z) = K
    c(z) = black
    return
  new z
  k(z) = K
  c(z) = red
  x = r_t(T)
  while true
    if K < k(x)
      if l(x) = null
        l(x) = z
        p(z) = x
        break
      x = l(x)
    else
      if r(x) = null
        r(x) = z
        p(z) = x
        break
      x = r(x)
```

```
if p(x) = r_t(T)
   return
{fix red black properties}
while c(p(z)) = red
   {cases L1 - L3}
   if p(z) = l(p(p(z)))
      y = r(p(p(z)))
      if c(y) = red {case L1}
         c(p(z)) = black
         c(y) = black
         c(p(p(z))) = red
         z = p(p(z))
      else {cases L2 and L3}
         if z = r(p(z)) {case L2}
            z = p(z)
            rotate_left(z)
         {case L3}
         c(p(z)) = black
         c(p(p(z))) = red
         rotate_right(p(p(z)))
   else if p(z) = r(p(p(z))) {cases R1 - R3}
      y = l(p(p(z)))
      if c(y) = red {case R1}
         c(p(z)) = black
         c(y) = black
         c(p(p(z))) = red
         z = p(p(z))
      else {cases R2 - R3}
         if z = l(p(z)) {case R2}
            z = p(z)
            rotate_right(z)
         {case R3}
         c(p(z)) = black
         c(p(p(z))) = red
         rotate_left(p(p(z)))
   c(r_t(T)) = black
```

1.2 Deleting

Deleting node z is actually replacing z with it's predecessor/successor N and fixing red black properties. If N is red when removed, then properties 1 - 5 still hold. If N is black, then properties 1, 2, 4 and for can be violated. Let x be the N's sole child (or sentinel). x is considered to have has an extra blackness received from N in a sense that this blackness has to be moved into some other node using rotations and recolorings. Let $w = r(p(x))$ be the sibling of x.

The following cases keep the same number of black nodes of all affected paths and fix properties 1 and 4 (property 2 is fixed later):

L1 w is red: colors of $p(x)$ and w are swapped, then $p(x)$ is left rotated; this case is reduced to cases 2, 3, 4.

L2 w is black, both w's children are black $\Rightarrow$ w is colored red, extra blackness of x is moved to $p(x)$;

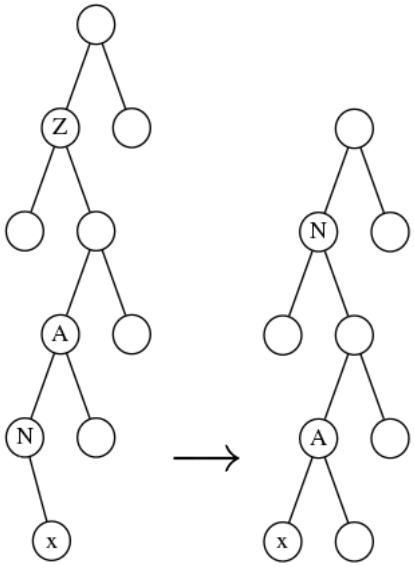

Figure 4: Deleting Z by replacing it with successor N

if $p(x)$ is red, then it is colored into black and the process finishes; otherwise the process is repeated on $p(x)$ which has extra blackness.

L3 w is black, $l(w)$ is red, $r(w)$ is black: colors of w and $l(w)$ are swapped and w is right rotated; thus, this case is reduced to case L4.

L4 w is black, $r(w)$ is red: w takes color of $p(x)$, $p(x)$ and $r(w)$ are colored black, $p(x)$ is left rotated, extra blackness of x is dropped making x colored black; this case finishes transformations.

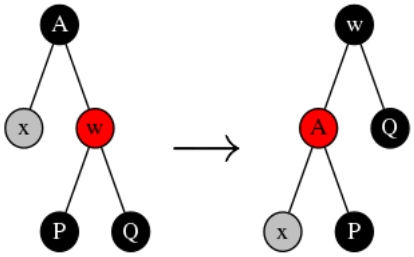

Figure 5: L1 case transformed to L2, L3, L4

Symmetric cases are for $x = r(p(x))$, $w = l(p(x))$:

R1 w is red: colors of $p(x)$ and w are swapped, then $p(x)$ is right rotated; this case is reduced to cases 2, 3, 4.

R2 w is black, both w's children are black $\Rightarrow$ w is colored red, extra blackness of x is moved to $p(x)$; if $p(x)$ is already red, then the process finishes, otherwise the process is repeated on $p(x)$.

R3 colors of w and p are swapped and w is left rotated.

R4 w is black, $l(w)$ is red: w takes color of $p(x)$, $p(x)$ and $l(w)$ are colored black, $p(x)$ is right rotated, extra blackness of x is dropped making x colored black; this case finishes transformations.

Complexity: $O(\lg n)$ **Input:** key K to delete
Output: none
remove(K)
  if $r_t(T) =$ **null**

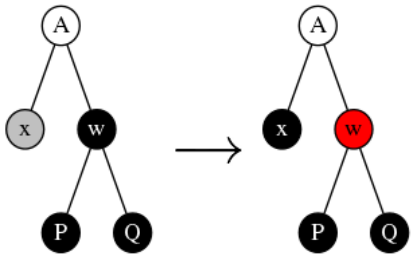

Figure 6: L2 case finishes or proceeds on $p(x)$

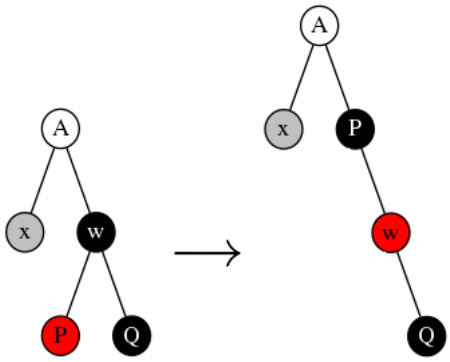

Figure 7: L3 case

return
$z = find(K)$
if $z =$ **null**
return
if $z = r_t(T)$
delete $r_t(T)$
return
if $c(z) =$ **red and** $l(z) = s(T)$ **and** $r(z) = s(T)$
if $z = l(p(z))$
$l(p(z)) = s(T)$
else if $z = r(p(z))$
$r(p(z)) = s(T)$
return
$N_s = successor(x), N_p = predecessor(x), N =$ **null**, $x =$ **null**
if $N_s =$ **null and** $N_p =$ **null**
$A = p(z)$
$N = A$
if $z = l(A)$
$l(A) = s(T)$
$x = l(A)$
else if $z = r(A)$
$r(A) = s(T)$
$x = r(A)$
else if $N_s \neq$ **null**
$N = N_s, A = p(N), k(z) = k(N_s)$
{*reconnect* A *with* N*'s right child*}
if $N_s = l(A)$

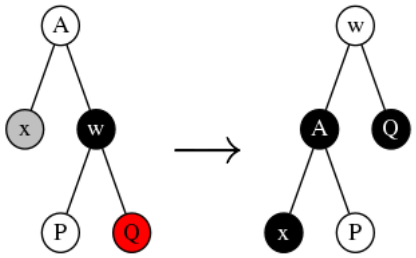

Figure 8: L4 case

```
        if r(N_s) ≠ null
            l(A) = r(N_s)
            p(r(N_s)) = A
        else
            l(A) = s(T)
        x = l(A)
    else if N_s = r(A)
        r(A) = s(T)
        x = r(A)
else if N_p ≠ null
    N = N_p, A = p(N), k(z) = k(N_p)
    {reconnect A with N's left child}
    if N_p = r(A)
        if l(N_p) ≠ null
            r(A) = l(N_p)
            p(l(N_p)) = A
        else
            r(A) = s(T)
        x = r(A)
    else if N_p = l(A)
        l(A) = s(T)
        x = l(A)
if c(N) = black or p(x) = N
    {fix red black properties}
    while x ≠ r_t(T) and c(x) = black
        if x = l(p(x)) {cases L1 - L4}
            w = r(p(x))
            if c(w) = red {case L1}
                c(w) = black
                c(p(x)) = red
                rotate_left(p(x))
                w = r(p(x))
            if c(w) = black and c(l(w)) = black and c(r(w)) = black {case L2}
                c(w) = red
                x = p(x)
                if c(x) = red
                    break
            else
                if c(r(w)) = black {case L3}
                    c(l(w)) = black
                    c(w) = red
```

```
            rotate_right(w)
            w = r(p(x))
          {case L4}
          c(w) = c(p(x))
          c(p(x)) = black
          c(r(w)) = black
          rotate_left(p(x))
          x = r_t(T) {break the loop}
      else if x = r(p(x)) {cases R1 - R4}
        w = l(p(x))
        if c(w) = red {case R1}
          c(w) = black
          c(p(x)) = red
          rotate_right(p(x))
          w = l(p(x))
        if c(w) = black and c(r(w)) = black and c(l(w)) = black {case R2}
          c(w) = red
          x = p(x)
          if c(x) = red
            break
        else
          if c(l(w)) = black {case R3}
            c(r(w)) = black
            c(w) = red
            rotate_left(w)
            w = l(p(x))
          {case R4}
          c(w) = c(p(x))
          c(p(x)) = black
          c(l(w)) = black
          rotate_right(p(x))
          x = r_t(T) {break the loop}
    c(x) = black
```

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Introduction to Algorithms; Second Edition

[2] Miodrag Zivkovic: Algoritmi

[3] Robert Sedgewick: Algorithms