Remove in red black tree.

Need to remove node v with ≤ 1 child.

If v is red, it is a leaf.  

Just remove it.  Done.
If $v$ has a red child, remove $v$ and color the child black.

$\Rightarrow$
OW, need to solve case of black node v with one black child (treat null as black).

Want to remove v.
Easy case: $v$ is root.

Just remove it to make one child the new root.
p: parent node of v
w: sibling of v
First, assume w is black.
Case I: w has a red child c

If c is closer to v than w, double rotate so
1) $c$ has children $p + w$

2) Color $c$ same as $p$ & then color $p$ black

3) Remove $v$.
Case 16:

1. Single rotate $w$ to be parent of $p+c$.
2. Give $w$ same color as $p$ then color $p$ black.
3) Remove v. Color w black (it's a new leader)

Case 2: w has only black children.
If $p$ is red

1) Color $p$ black.

2) Remove $v$.

3) Color $w$ red.
If $p$ is black...

Create a new dummy node $v'$, $q$'s parent instead.
2) Color

w red.

3) Remove

w.

4) Remove

v using above rules.
(in practice, we removed $v$ already & are maintaining a link to the parent of the imaginary node we're trying to remove)
But what if w is red?

\[ \Rightarrow p \text{ is black} \]

1) Rotate w so it has child p.
2) Color w black & p red.

3) See previous cases. (at most one step!)
AVL vs. red-black

AVL trees $\leq 1.44 \lg n$

red-black $\leq 2 \lg n$

AVL is faster with many contains.

Fewer rebalances for red-black tree.

Better for many inserts.
removes.

Everybody big uses red-black trees with parent pointers (Java Collections, libcpp, etc.)
Example:

Binary counter

Counter \([0 \ldots J]\)

\(\text{bit } i = 1 \text{ iff represented number has } 2^i \) in its unique rep as a sum of powers of 2.
public static increment(boolean[] counter) {
    int i = 0;
    while (counter[i]) {
        counter[i] = false;
        i++;
    }
    counter[i] = true;
}

Θ(k) time to flip k bits.

n uses (1 + \log_2 n) + 1 bits.
increment for 0 to n
takes $\Theta(\log n)$ time in worst-case.

\[
\Rightarrow O(n \log n) \text{ time to increment } v \text{ from } 0 \text{ } n \text{ times.}
\]

Want to know an amortized time bound for each
increment.

Formally, you can use any amortized time for each operation, but they must sum to at least sum of actual times. I may say cost instead of time.
Goal: An amortized bound on the 6 flips in one increment, as we go 0 to n.
Summation.

Bit $i$ flips every $2^i$ increments.

So bit $i$ flips $\lfloor \sqrt[2^i]{n} \rfloor$ times.

In total

$$\sum_{\dot{\omega} = 0}^{\dot{\omega} = \infty} \left( \frac{n}{2^i} \right) < \sum_{\dot{\omega} = 0}^{\dot{\omega} = \infty} \frac{n}{2^\dot{\omega}} = 2^n$$
So we have

\[ \frac{2n}{n} = 2 \] bits per increment.
Taxation

IRS charges $2 for each increment.

Immediately spend $1 to flip a bit to 1. Other $1 pays to flip it back to 0 later.
Every operation paid for so amortized cost is 2.

- or -

Bit \( i \) pays $\frac{1}{2^i}$ with every increment.

Each bit will pay all between consecutive flips.
Total tax per inc. is $2.
Charging

Charge to operations in the past.

Amortized cost is actual cost - what you charged + what was charged to you.
If an increment flips $k$ bits, charge each 1→0 flip to the last operation that set the bit to 1.

\[ \Rightarrow \text{were charged } \leq 1. \]

Amortized cost is $\leq k - (k-1) + 1 = 2$.
Potential

\( \Phi_i \): potential after \( i \) operations

\( c_i \): actual cost of \( i \)th operation

Define amortized cost as

\[ a_i = c_i + \Phi_i - \Phi_{i-1} \]
Suppose we do $m$ operations. Total amortized cost is

$$m \sum_{\bar{\imath} = 1}^m \Xi a_{\bar{\imath}} = \sum_{\bar{\imath} = 1}^m \Xi \left( c_{\bar{\imath}} + \varphi_{\bar{\imath}} - \varphi_{\bar{\imath}}^0 \right)$$

Call a potential valid if $\varphi_{\bar{\imath}} - \varphi_{\bar{\imath}}^0 \geq 0$ for all $\bar{\imath}$.
Valid \Rightarrow \sum_{i=1}^{m} E c_i = -(\Phi_m - \Phi_0) + \sum_{i=1}^{m} \sum_{j=1}^{n} a_i j_i \\
increment \Phi_i = \# 1s \ in \ rep. \ of \ i
\[ a_{\phi, \omega} = \# \text{6 bits 0} \Rightarrow 1 + (c_{\phi, \omega}) \]
\[ \# \text{6 bits 1} \Rightarrow 0 + \]
\[ \# \text{6 bits 0} \Rightarrow 1 - (\phi_{\phi, \omega}) \]
\[ \# \text{6 bits 1} \Rightarrow 0 \]
\[ = 2 \cdot (\# \text{6 bits 0} \Rightarrow 1) \]
\[ = 2 \]