Red-black tree remove

Down to remove node with one child.

If v is red, \( \rightarrow \) No children.
Just remove it.
O.W.

If \( v \) has a red child,

promoted \( c \) as a new black node.

O.W. what to do is \( v \) has one black child (may be null)
If $v$ is the root, just remove $v$ and make its child the root.

If $v$ has red sibling

rotate so $w$ has child $p$. Color $w$ black & $p$ red.
Now to case: v has a black sibling

- If w has a red child, c
  - If c is closer to v...
    - Double rotate so c is parent of p & w.
    - Give c p's color.
p is black

Remove v

FS o is the farthest child
Single rotate so w is on top with children cop.

Color w as p is colored,
color p black
remove v
If \( w \) has no red child

If \( p \) is black, add new black node \( v' \) above \( p \) with \( p \) its only child, remove \( y \) color \( w \) red

Recursively remove \( v' \)
If v was red, just make p black, w red, remove v.
2-3-4 tree remove

If leaf has 2 or 3 elements, just remove it.

O.w., how to remove an empty one-child node somewhere in tree.
If root, just remove.

0.w.

Find adjacent child.

If 2 or 3 elements, promote closest element to parent node.
demote element that was separating empty node from sibling

If sibling has one element,
steal from parent & fuse with sibling

skip list
Want to sort elements

\[
\text{Merge Sort } (A[0,...,n-1])
\]

if \( m \geq 2 \) then

\[
m \leftarrow \lfloor n/2 \rfloor
\]

\[
\text{Merge Sort } (A[0,...,m])
\]

\[
\text{Merge Sort } (A[m+1,...,n-1])
\]

\[
\text{Merge } (A[0..n-1], m) \leftarrow O(n)
\]

\( T(n) \): worst-case time for Merge Sort on \( n \) elements

\[
T(0) = O(1)
\]

\[
T(1) = O(1)
\]

\( \text{w.t.} \)

\[
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)
\]

\[
= 2T(\lfloor n/2 \rfloor) + O(n)
\]
\[ T(n) = O(n \log n) \]
(can prove with induction)
(how to solve \( T(n) \) won't be on mid term)

\[ \text{Strange Recurrence:} \]
\[ S_r(i,j) = \begin{cases} 
  9 & \text{if } i = 0 \\
  2 & \text{if } i > 0 \\
  S_R(i-1,j) + S_R(i,j-1) & \text{else}
\end{cases} \]

\[ T(m,n) = \text{time to solve } S_R(m,n) \text{ recursively} \]
\[ T(m,n) = T(m-1,n) + T(m-1,n-1) + O(1) \text{ for } m,n \]
Proof by induction:

Assume for all \(0 \leq k < n\),
\[ \sum_{i=0}^{k} 3^i = 3^{k+1} - 1 \]
and
\[ \sum_{i=0}^{k} 3^i = 3^{k+1} - 1 \]

If \(n = 0\),
\[ \sum_{i=0}^{n} 3^i = 3^0 = 1 = \frac{3^1 - 1}{2} \]

If \(n \geq 1\),
\[ \sum_{i=0}^{n} 3^i = \sum_{i=0}^{n-1} 3^i + 3^n \]
\[ = 3 + \sum_{i=0}^{n-1} 3^i \]
\[ = 3 + \frac{3^n - 1}{2} \]
\[ = \frac{2 \cdot 3^n + 3^n - 1}{2} \]
For all cases, sum is \( \frac{3^{n+1} - 1}{2} \).
How many nodes in $\mathcal{F}_n$?

Claim: $\mathcal{F}_n$ has $F_{n+2} - 1$ nodes.

Proof: Assume $\mathcal{F}_k$ has $F_{k+2} - 1$ nodes for any $1 \leq k < n$.

If $n = 1$, $\mathcal{F}_0$ nodes is

\[ 1 = F_3 - 1 = F_{n+2} - 1 \]

If $n \geq 2$

From (6), $F_n$ has $F_n$ leaves.

$\Rightarrow$ $\mathcal{F}_n$ has ($\#$ leaves in $\mathcal{F}_{n-1}$) + $F_n$ nodes
\[ F_{n+1} = F_n + F_{n+2} - 1 \]