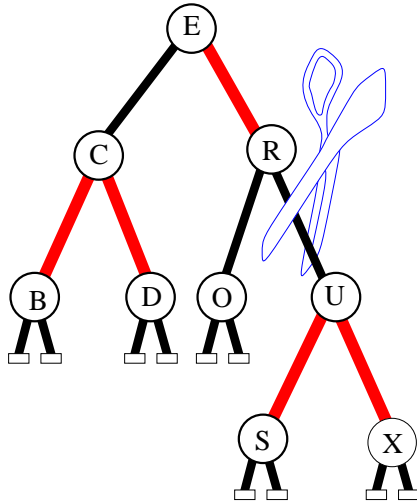


## Deletion from Red-Black Trees



ern

12.235

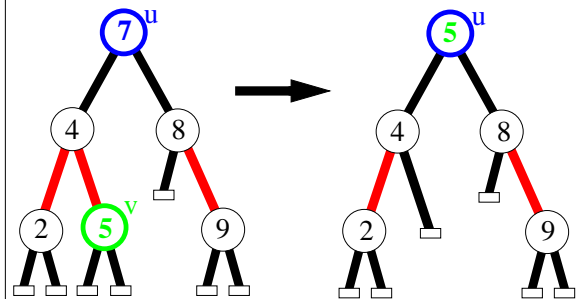


## Setting Up Deletion

As with binary search trees, we can always delete a node that has at least one external child

If the key to be deleted is stored at a node that has no external children, we move there the key of its inorder predecessor (or successor), and delete that node instead

**Example:** to delete key 7, we move key 5 to node u, and delete node v



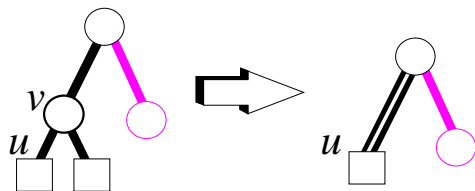
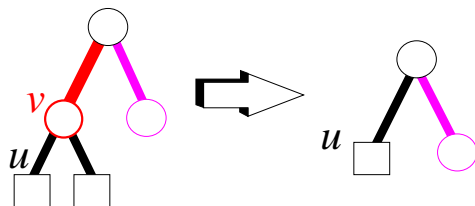
ern

12.236



## Deletion Algorithm

1. Remove  $v$  with a `removeAboveExternal` operation
2. If  $v$  was **red**, color  $u$  black. Else, color  $u$  **double black**.



3. While a **double black** edge exists, perform one of the following actions ...

ern

12.237



## How to Eliminate the Double Black Edge

- The intuitive idea is to perform a “**color compensation**”
- Find a red edge nearby, and change the pair ( **red**, **double black** ) into ( **black**, **black** )
- As for insertion, we have two cases:
  - **restructuring**, and
  - **recoloring (demotion)**, inverse of promotion)
- Restructuring resolves the problem locally, while **recoloring** may propagate it two levels up
- Slightly more complicated than insertion, since two restructurings may occur (instead of just one)

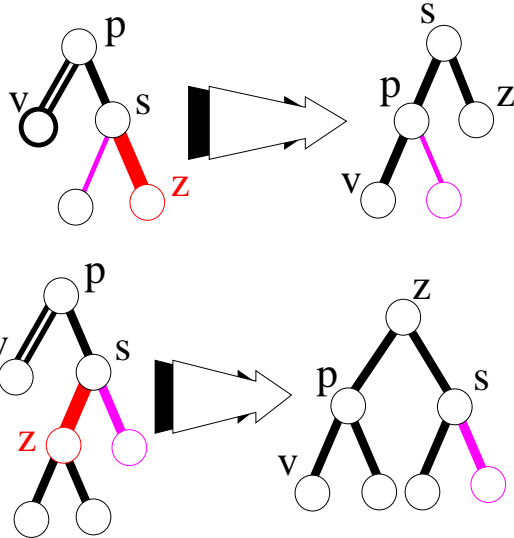
ern

12.238



## Case 1: black sibling with a red child

- If sibling is **black** and one of its children is **red**, perform a *restructuring*

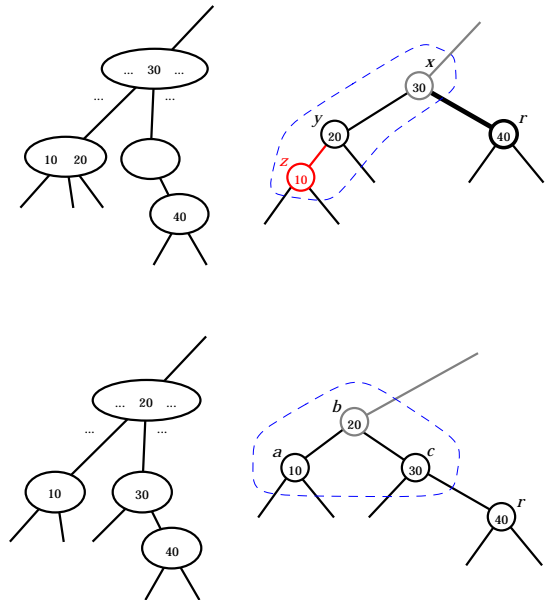


erm

12.239



## (2,4) Tree Interpretation



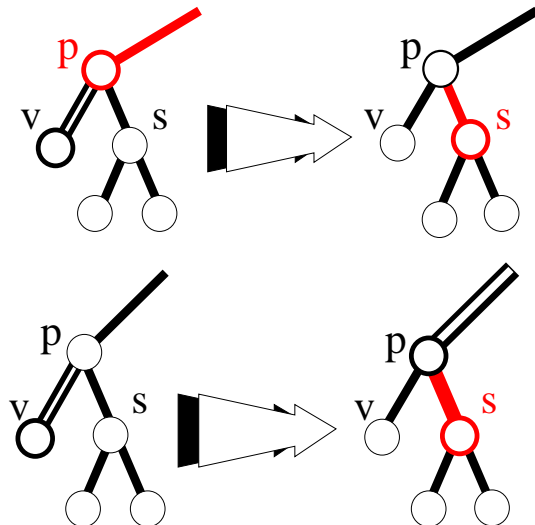
erm

12.240



## Case 2: black sibling with black children

- If sibling and its children are **black**, perform a *recoloring*
- If parent becomes **double black**, *continue* upward

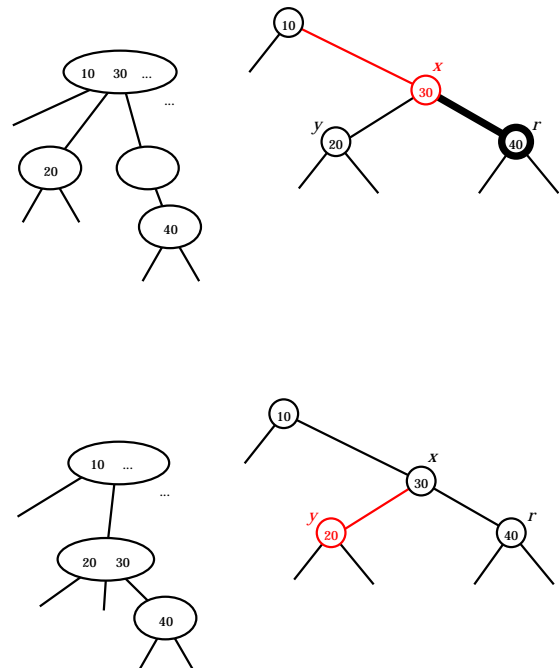


erm

12.241



## (2,4) Tree Interpretation



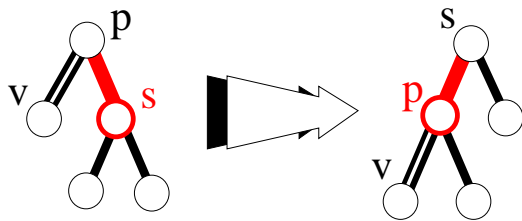
erm

12.242



## Case 3: red sibling

- If sibling is red, perform an *adjustment*
- Now the sibling is **black** and one the of previous cases applies
- If the next case is recoloring, there is no propagation upward (parent is now **red**)



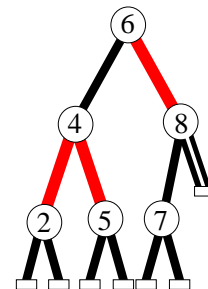
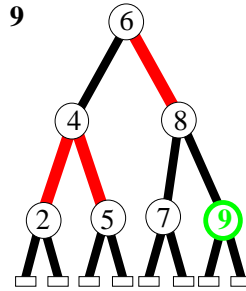
erm

12.243



## How About an Example?

Remove 9



erm

12.244



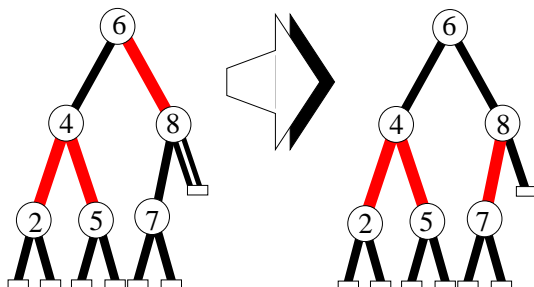
## Example

What do we know?

- Sibling is black with black children

What do we do?

- Recoloring



erm

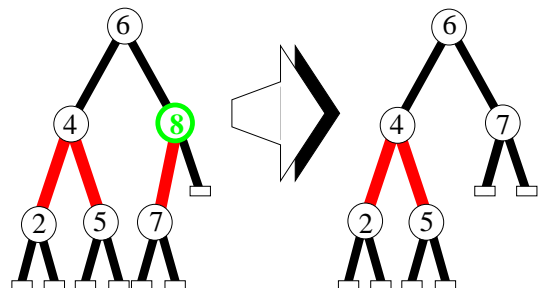
12.245



## Example

Delete 8

- no double black



erm

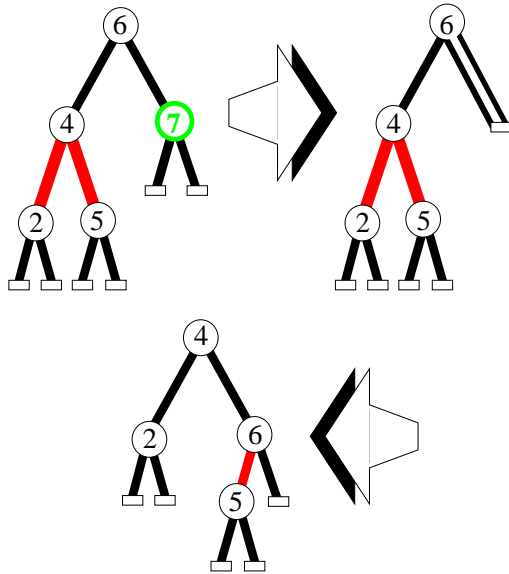
12.246



## Example

### Delete 7

- Restructuring

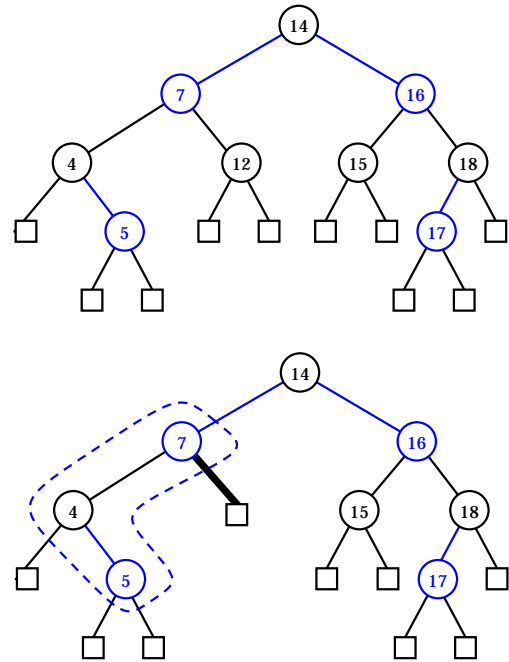


erm

12.247



## Example

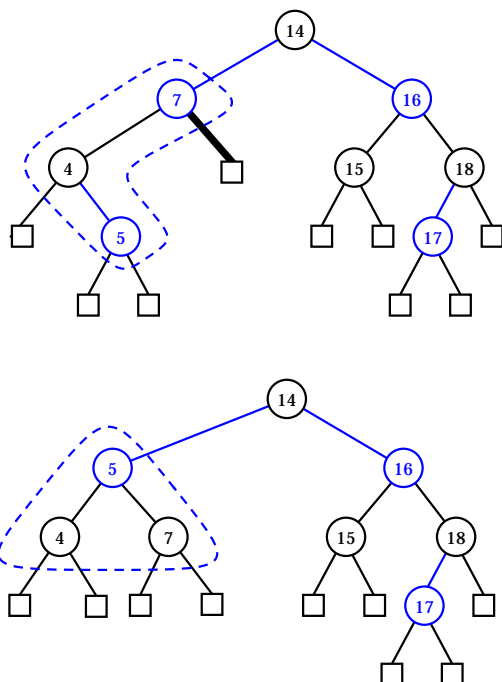


erm

12.248



## Example



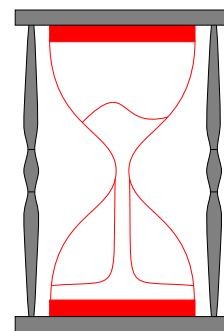
erm

12.249



## Time Complexity of Deletion

Take a guess at the time complexity of deletion in a **red**-black tree . . .



erm

12.250



# $O(\log N)$

What else could it be?!



## Colors and Weights

Color	Weight
red	0
black	1
double black	2



Every root-to-leaf path has the same weight

There are no two consecutive **red** edges

- Therefore, the length of any root-to-leaf path is at most twice the weight

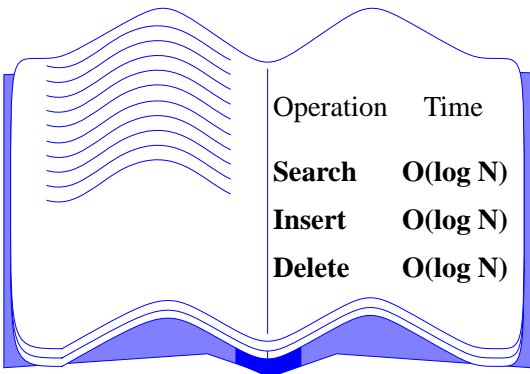


## Bottom-Up Rebalancing of Red-Black Trees

- An insertion or deletion may cause a local *perturbation* (two consecutive **red** edges, or a **double-black** edge)
- The perturbation is either
  - *resolved locally* (restructuring), or
  - *propagated* to a higher level in the tree by recoloring (promotion or demotion)
- $O(1)$  time for a restructuring or recoloring
- At most one restructuring per insertion, and at most two restructurings per deletion
- $O(\log N)$  recolorings
- Total time:  $O(\log N)$



## Red-Black Trees



Operation	Time
Search	$O(\log N)$
Insert	$O(\log N)$
Delete	$O(\log N)$

