Deletion from Red-Black Trees
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.

![Diagram showing the deletion process]
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 ...
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)
Case 1: black sibling with a red child

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

---

**Diagram:**

- Initial configuration with a black sibling and a red child.
- After restructuring, the sibling becomes red and the child black.
(2,4) Tree Interpretation
Case 2: black sibling with black children

- If sibling and its children are black, perform a *recoloring*
- If parent becomes *double black*, *continue* upward
(2,4) Tree Interpretation
Case 3: red sibling

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

Remove 9

Remove 9 from the Red-Black Tree.
**Example**

**What do we know?**
- Sibling is black with black children

**What do we do?**
- Recoloring
Delete 8
- no double black
Example

Delete 7

- Restructuring
Example
Example
Time Complexity of Deletion

Take a guess at the time complexity of deletion in a red-black tree . . .
What else could it be?!
Colors and Weights

<table>
<thead>
<tr>
<th>Color</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>0</td>
</tr>
<tr>
<td>black</td>
<td>1</td>
</tr>
<tr>
<td>double black</td>
<td>2</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(\log N)$</td>
</tr>
</tbody>
</table>