Red-Black Tree Deletion (Case 3)

This is the last case where we have to deal with a double black node r. In this situation, y, the sibling of r, is red. If y is a right child of x (where x is r's parent), let z be the right child of y. Otherwise, let z be the left child of y. (Note that both x and z must be black.) Perform a restructuring on the node z, placing y where x used to be:

```
        x                                   y
      /   \                               /   \
    y      rdb          ➔               z       x
   /                                              \
  z                                                rdb
```

Since y's right child in the original picture MUST BE black, x's new left child MUST BE black as well. This puts us in either case 1 or case 2 to deal with the "double black" node.

Here is the symmetric situation:

```
        x                                   y
      /   \                               /   \
   rdb      y           ➔               x       z
              \                        /
               z                    rdb
```

Here we have that y's original left child must have been black, thus, in the restructured picture, x's new right child must be black as well, putting us in either case 1 or case 2 as desired.

In both of these situations, since x is red, we will NOT propagate the "double black" node. Instead it will be handled in one operation of either case 1 or case 3.
Consider deleting 15 from the Red-Black Tree below:

```
          10                                  10
        /     \                             /     \
       5       20                          5       20
     /   \     / \          ➔            /   \    /   \
    3     7   15  25                    3     7  ndb   25
   / \   / \                           / \   / \
  1   4 6   8                         1   4 6   8
```

This is actually case 2. Here we will recolor as follows:

```
          10
        /     \
       5       20db
     /   \     / \
    3     7   n   25
   / \   / \
  1   4 6   8
```

Now we have a situation where we are in case 3. Here we will have x=10, y=5, and z=3.

```
          5
        /     \
      3        10
     / \      /   \
    1   4    7     20db
            / \   /   \
           6   8 n     25
```

Now, we are in case 2, since both of 7's children are black. Deal with this case with a recoloring:

```
          5
       /      \
     3        10
   /  \      /   \
  1    4   7    20
          /  \   /   \
          6  8  n    25
```

Consider the corresponding delete in a 2-4 Tree:

```
          5,10
        /   |   \
     3     7     20
   /  \   / \   / \
  1    4  6  8 15  25
```

```
          5,10
        /   |   \
     3     7     20, which causes a fusion operation
   /  \   / \   / \
  1    4  6  8      25
```

```
          5,10
        /   |   \
     3     7     20, now, to remedy this, drop 20 into the child.
   /  \   / \    |     corresponding to the first case 2 work.
  1    4  6  8    25
```

```
          5,10
        /    |    \
     3      7         , now, fuse again
   /  \    / \     |    this is what corresponds to case 3.
 1    4  6  8  20, 25
```

```
          5,10
        /    |
     3      7         , to accommodate, drop 10 into fused node
   /  \    / \     |    this is analogous to the last case 2 we see.
 1    4  6  8  20, 25
```

```
          5
        /    \
     3      7,10
   /  \    / |     \
 1    4  6  8  20, 25
```

Notice that case three corresponds to the case when we have to drop a value from a parent node into a child after a fusion, where the parent node DOES have a value to spare. This is why after this occurs, only a single application of case 1 or case 2 is necessary. The difference in case corresponds to a difference in which value from the parent node will end up dropping into the fused node.

Summarizing Red-Black Tree Delete

We have a couple simple cases to deal with, which we can do without any extra work. These correspond to removing a value from a 2-4 tree where there are other values at the node the removal is made form.

The rest of the cases result in a "double black" colored node. These cases all correspond to restructuring operations in the 2-4 Tree, such as fusions, and dropping elements into a node. The goal is to deal with the double black(DB) node to get rid of this property. Here is the outline of these cases:

Case 1: <u>DB node has black sibling with at least one red child.</u>

> **This fixes a the problem structurally. No extra work is required after this case completes. This corresponds to a transfer operation in a 2-4 Tree.**

Case 2: <u>DB node has black sibling with two black children.</u>

> **This uses a recoloring and no structural change. It may solve the problem, but may ALSO propagate the DB node to the parent of the current DB node. This corresponds to a fusion and drop operation in a 2-4 Tree.**

Case 3: <u>DB Node has red sibling</u>

> **A structural change here puts you in case 1 or case 2. At this point, a single application of either case is sufficient. This corresponds to a fusion where you have enough values in the parent node to drop one into the fused child.**