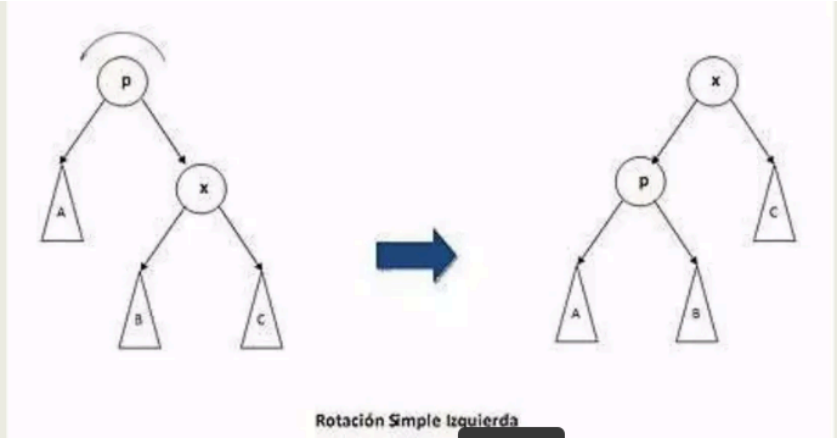
ÁRBOLES ROJINEGROS(ELIMINACIÓN)

Ericka Araya-C20553

Este trabajo ayuda a entender mejor los casos que contemplan la operación de RB-Delete-Fixup(T,x) la cual se utiliza para restablecer las propiedades 1, 2, y 4 de la estructura de datos Árbol Rojinegro al realizar la operación RB-Delete(T,z), que se encarga de eliminar un nodo del árbol.

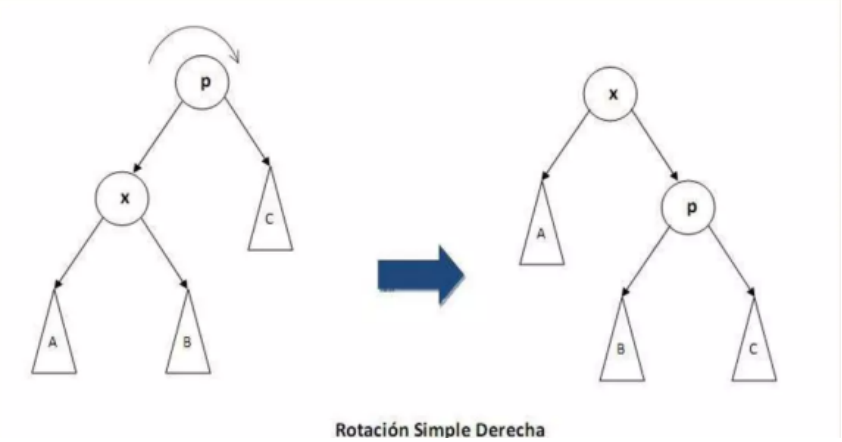
Para entender estos casos, es importante entender que son las rotaciones, ya que estas son necesarias para que cumpla las propiedades de un árbol rojo-negro, ya que en los casos como inserción y eliminación al final hay ocasiones que se necesita reestructurar el árbol, por lo cual existen la rotación: izquierda y derecha.

- **Rotación izquierda:** Se trata de mover un nodo del hijo derecho hacia la posición del padre, mientras que el padre se mueve hacia la posición como hijo izquierdo del nodo que se movió. A continuación, el hijo izquierdo del nodo que se movió se convierte en el hijo derecho del esposo original.



Rotación Simple Izquierda

- **Rotación derecha:** Es necesario mover un nodo del hijo izquierdo hacia la posición del padre, mientras que el padre se mueve hacia la posición como hijo izquierdo del nodo que se movió. En última instancia, el hijo derecho del nodo que se movió se convierte en el hijo izquierdo del padre original.



Rotación Simple Derecha

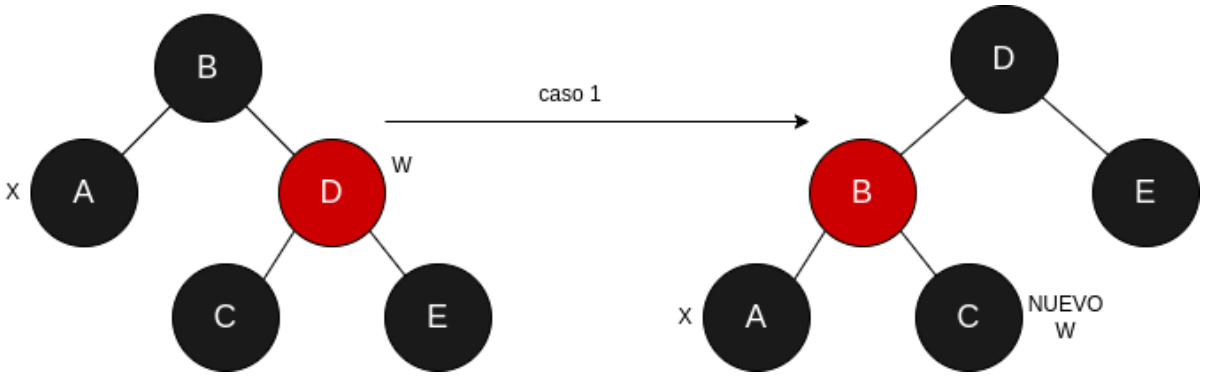
Comprendiendo mejor las rotaciones podemos ver con más claridad los casos que contemplan la operación de RB-Delete-Fixup(T,x), nos guiaremos del siguiente pseudocódigo:

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left                    // is x a left child?
 3          w = x.p.right                   // w is x's sibling
 4          if w.color == RED
 5              w.color = BLACK
 6              x.p.color = RED
 7              LEFT-ROTATE(T, x.p)                          } case 1
 8              w = x.p.right
 9          if w.left.color == BLACK and w.right.color == BLACK
10              w.color = RED                                } case 2
11              x = x.p
12          else
13              if w.right.color == BLACK
14                  w.left.color = BLACK
15                  w.color = RED
16                  RIGHT-ROTATE(T, w)                       } case 3
17                  w = x.p.right
18              w.color = x.p.color
19              x.p.color = BLACK
20              w.right.color = BLACK                        } case 4
21              LEFT-ROTATE(T, x.p)
22              x = T.root
23      else // same as lines 3–22, but with "right" and "left" exchanged
24          w = x.p.left
25          if w.color == RED
26              w.color = BLACK
27              x.p.color = RED
28              RIGHT-ROTATE(T, x.p)
29              w = x.p.left
30          if w.right.color == BLACK and w.left.color == BLACK
31              w.color = RED
32              x = x.p
33          else
34              if w.left.color == BLACK
35                  w.right.color = BLACK
36                  w.color = RED
37                  LEFT-ROTATE(T, w)
38                  w = x.p.left
39              w.color = x.p.color
40              x.p.color = BLACK
41              w.left.color = BLACK
42              RIGHT-ROTATE(T, x.p)
43              x = T.root
44  x.color = BLACK
```

Caso 1: El hermano(w) de x es rojo, por lo cual su condición es que: w.color == RED, si esto se cumple se deben realizar las siguiente acciones:

- Cambia el color de w a negro.
- Cambia el color de x.p a rojo.
- Realiza una rotación a la izquierda en x.p.

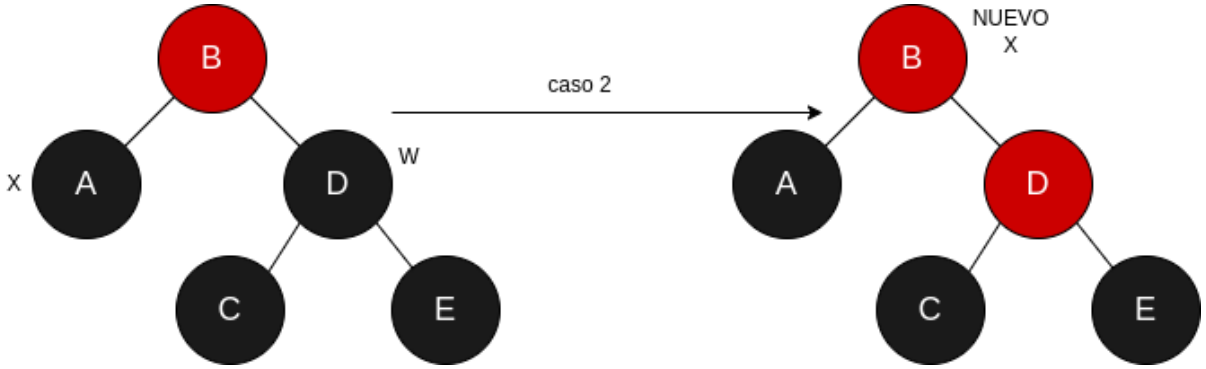
La idea es convertir w en negro y mover x.p a una posición donde w sea el hermano de x, que es negro, lo que facilita el manejo en los siguientes casos.
Ejemplo:



Caso 2: El hermano(w) de x es negro y ambos hijos de w son negros. Sus condiciones son que w.color == BLACK, esta no se agrega al código, ya que si no se cumple la condición en el caso 1, por lógica se sabe que entonces w.color va a ser BLACK, luego se debe cumplir w.left:color == BLACK y w.right:color == BLACK. Si se cumple entonces se realiza:

- Cambio del color de w a rojo.
- Mueve x a x.p.

El objetivo es propagar el "doble negro" hacia arriba en el árbol, lo que permite continuar la corrección en niveles superiores del árbol.
Ejemplo:

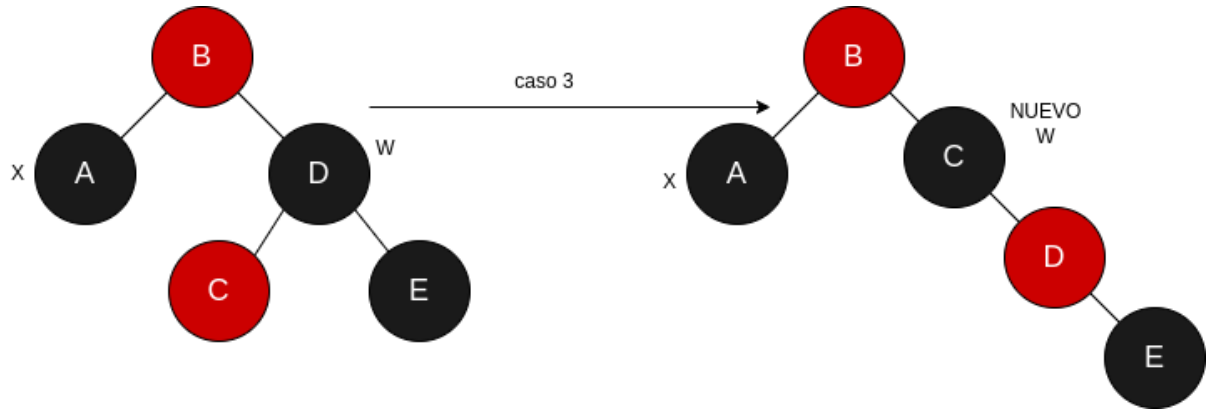


Caso 3: El hermano(w) de x es negro, el hijo izquierdo de w es rojo y el hijo derecho de w es negro, cumpliendo las condiciones de: w.color == BLACK, w.left:color == RED, y w.right.color == BLACK, luego realizando las acciones de:

- Cambia el color de w.left a negro.
- Cambia el color de w a rojo.

- Realiza una rotación a la derecha en w.

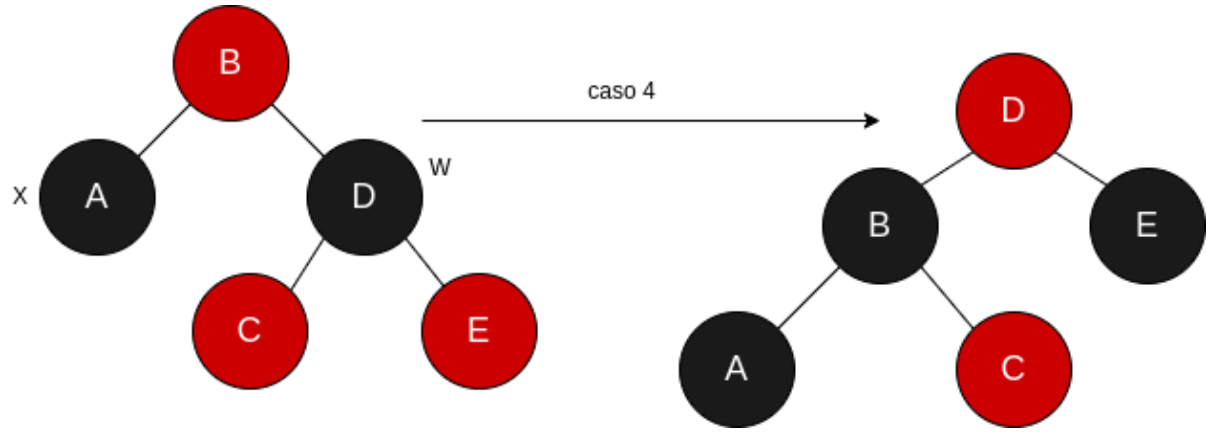
Con esto se transforma w en un nodo con un hijo rojo derecho, lo que facilita la resolución en el siguiente y último caso.
Ejemplo:



Caso 4: El hermano(w) de x es negro y el hijo derecho de w es rojo, o sea cumpliendo la condición de que w.color == BLACK y w.right:color == RED, si se cumple entonces:

- Cambia el color de w al color de x:p.
- Cambia el color de x.p a negro.
- Cambia el color de w.right a negro.
- Realiza una rotación a la izquierda en x.p.
- Mueve x a la raíz.

Esto para eliminar el "doble negro" y restaurar las propiedades del árbol rojo-negro.
Ejemplo:



Conclusión

La función RB-DELETE-FIXUP(T, x) es fundamental para mantener la integridad y las propiedades de los árboles rojo-negro tras una eliminación. Al manejar cuidadosamente los diferentes casos, el algoritmo asegura que todas las propiedades del árbol se restablezcan, permitiendo que el árbol siga siendo una estructura de datos eficiente y balanceada para operaciones de búsqueda, inserción y eliminación.