**Red Black Tree-**

Red-Black tree is a binary search tree in which every node is colored with either red or black. It is a type of self balancing binary search tree. It has a good efficient worst case running time complexity.

* **To add an element to a Red Black Tree, we must follow this algorithm:**
  + 1) Check whether tree is Empty.
  + 2) If tree is Empty then insert the newNode as Root node with color Black and exit from the operation.
  + 3) If tree is not Empty then insert the newNode as a leaf node with Red color.
  + 4) If the parent of newNode is Black then exit from the operation.
  + 5) If the parent of newNode is Red then check the color of parent node's sibling of newNode.
  + 6) If it is Black or NULL node then make a suitable Rotation and Recolor it.
  + 7) If it is Red colored node then perform Recolor and Recheck it. Repeat the same until tree becomes Red Black Tree.

1) Kiểm tra xem cây có trống không.

2) Nếu cây trống thì hãy chèn newNode làm nút gốc với màu Đen và thoát khỏi thao tác.

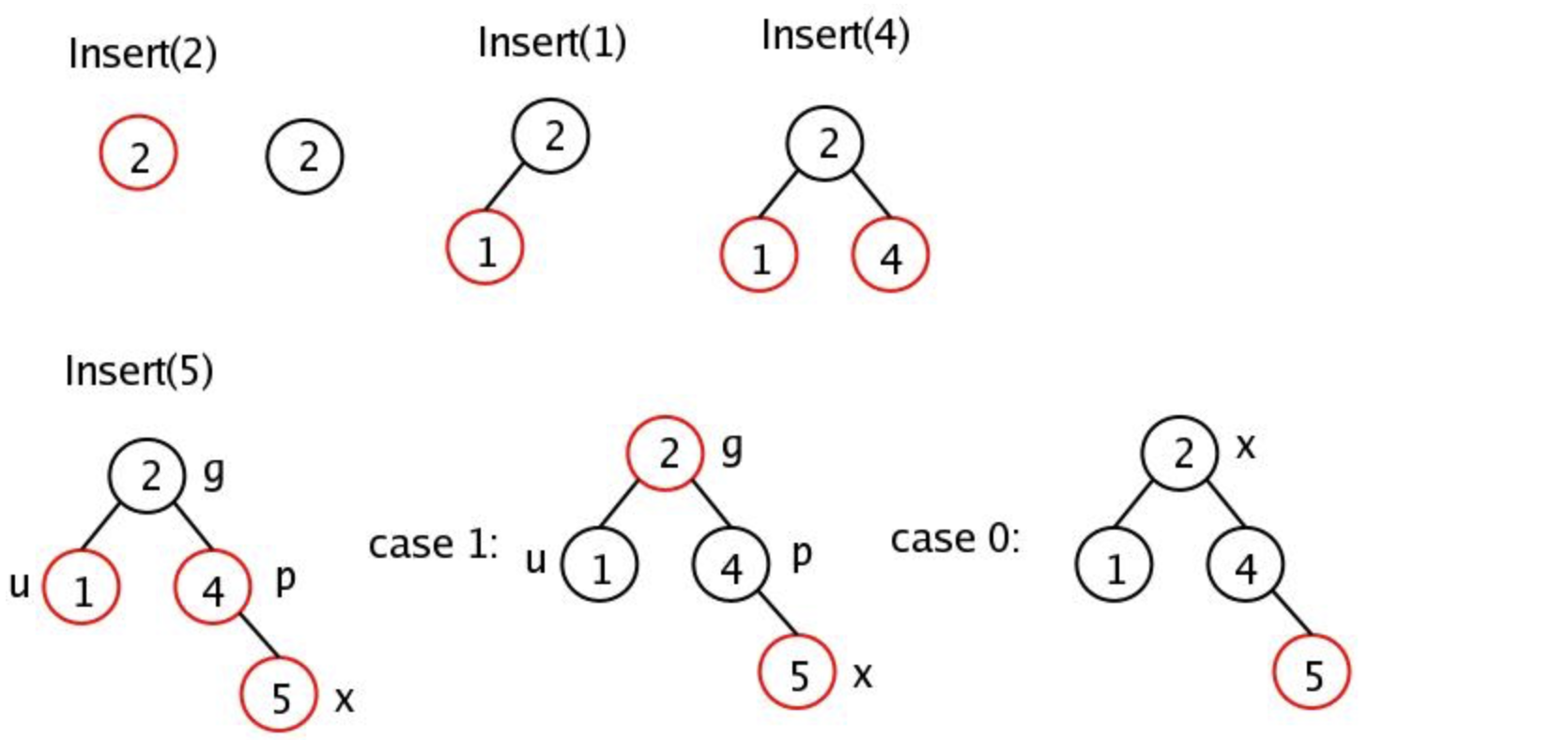
3) Nếu cây không trống thì hãy chèn newNode làm nút lá có màu Đỏ.

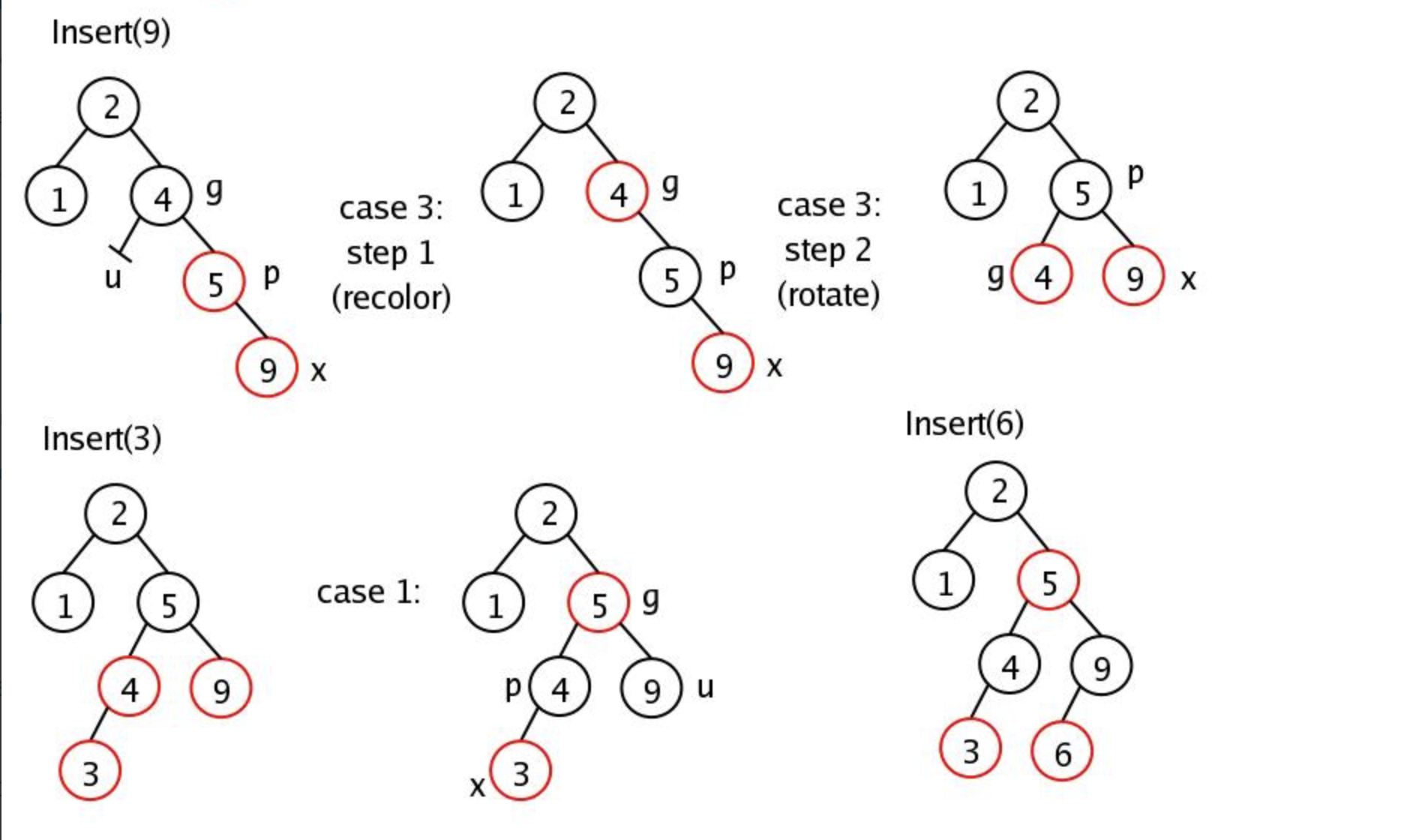
4) Nếu cha mẹ của newNode là Đen thì hãy thoát khỏi hoạt động.

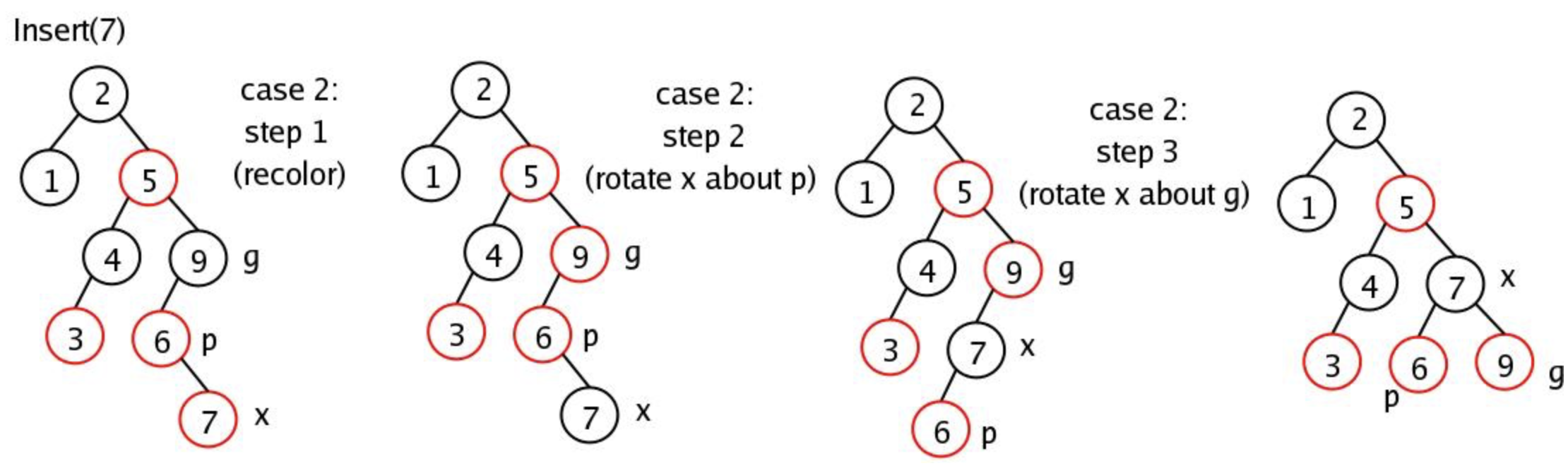
5) Nếu cha mẹ của newNode là Red thì hãy kiểm tra màu của anh chị em ruột của newNode.

6) Nếu nó là nút Đen hoặc NULL thì hãy tạo một Vòng quay phù hợp và Đổi màu nó.

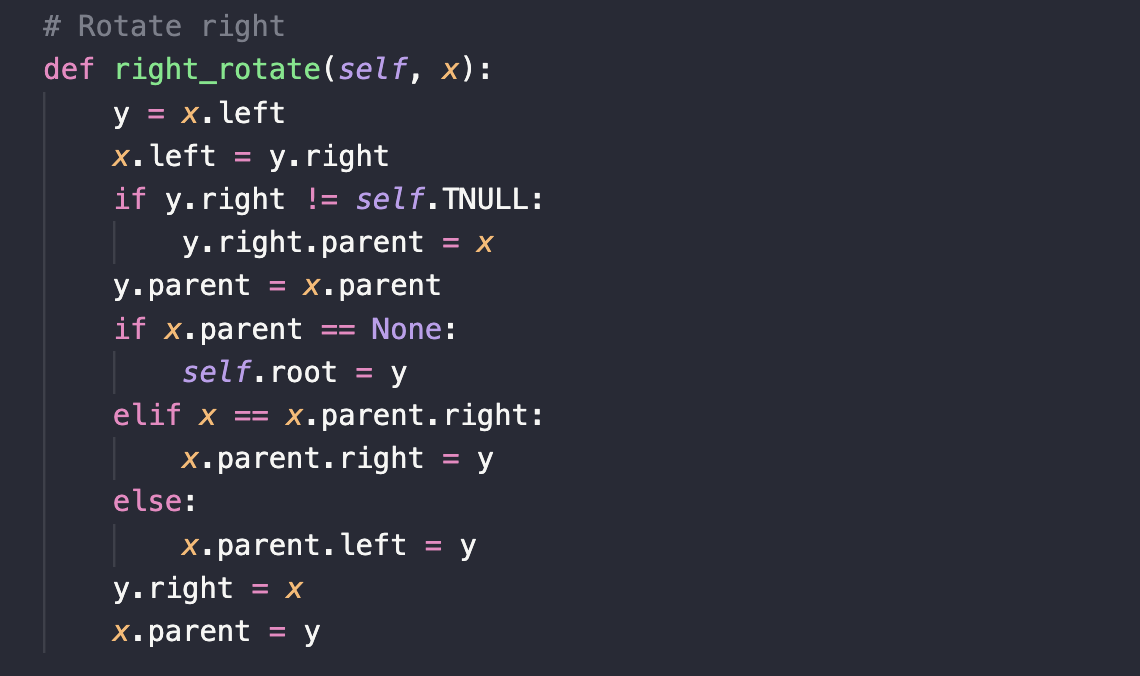
7) Nếu nó là nút màu đỏ thì hãy thực hiện Tô màu lại và Kiểm tra lại nó. Lặp lại tương tự cho đến khi cây trở thành Cây Đỏ Đen.



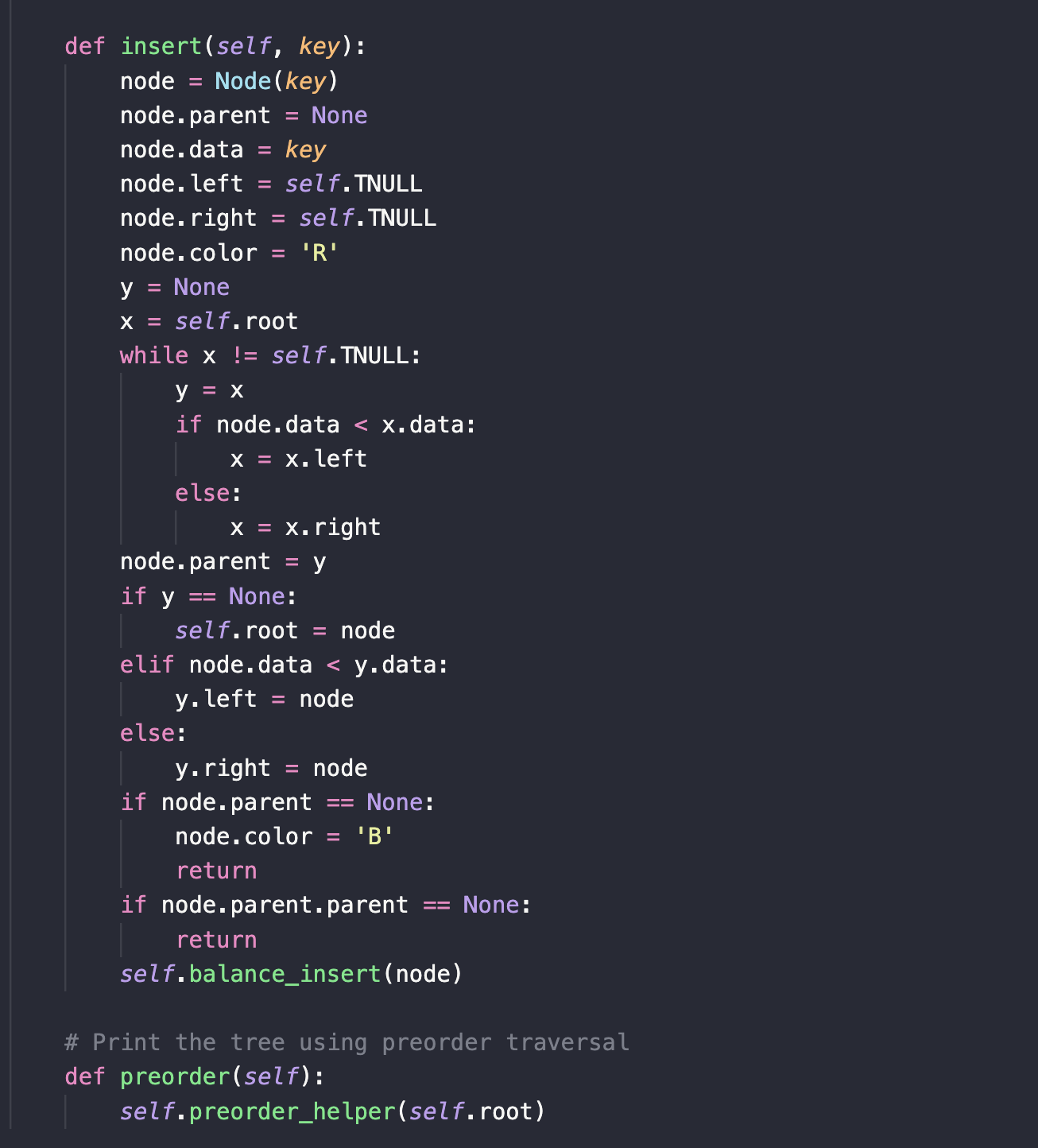














**Algorithm**

* **Basic operations associated with Red Black Tree:**

* + **Deletion of a node in Red Black Tree:**

**1)** Perform standard Binary Search Tree delete. When we perform standard delete operation in BST, we always end up deleting a node which is either leaf or has only one child (For an internal node, we copy the successor and then recursively call delete for successor, successor is always a leaf node or a node with one child). So we only need to handle cases where a node is leaf or has one child. Let v be the node to be deleted and u be the child that replaces v (Note that u is NULL when v is a leaf and color of NULL is considered as Black).

**2)** Simple Case: If either u or v is red, we mark the replaced child as black (No change in black height). Note that both u and v cannot be red as v is parent of u and two consecutive reds are not allowed in red-black tree.

**3)** If Both u and v are Black.   
  
**3.1)** Color u as double black. Now our task reduces to convert this double black to single black. Note that If v is leaf, then u is NULL and color of NULL is considered as black. So the deletion of a black leaf also causes a double black.   
  
**3.2)** Do following while the current node u is double black and it is not root. Let sibling of node be s.  
(a): If sibling s is black and at least one of sibling’s children is red, perform rotation(s). Let the red child of s be r. This case can be divided in four subcases depending upon positions of s and r.

**(i)** Left Left Case (s is left child of its parent and r is left child of s or both children of s are red). This is mirror of right right case shown in below diagram.

**(ii)** Left Right Case (s is left child of its parent and r is right child). This is mirror of right left case shown in below diagram.

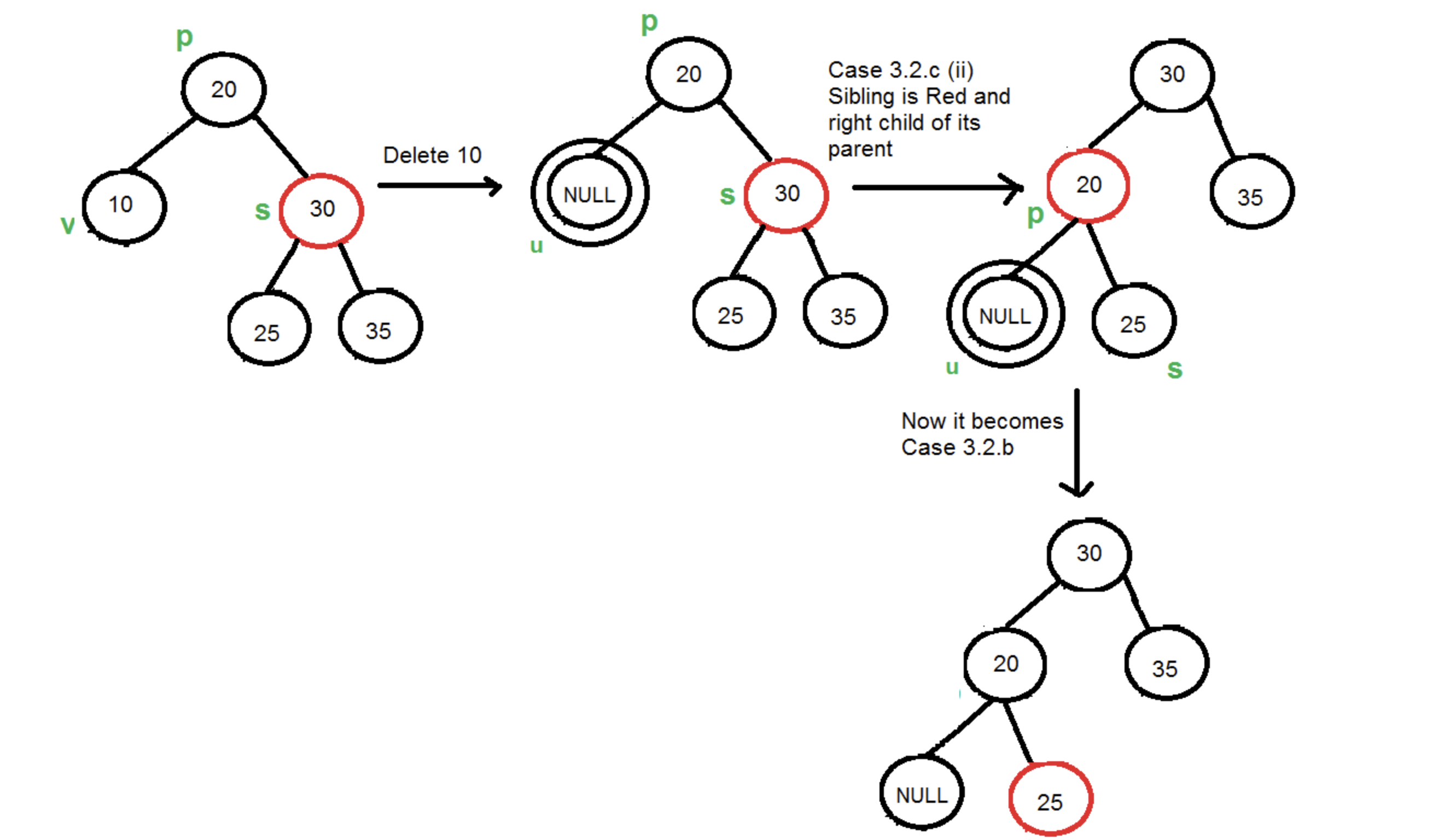
**(iii)** Right Right Case (s is right child of its parent and r is right child of s or both children of s are red)

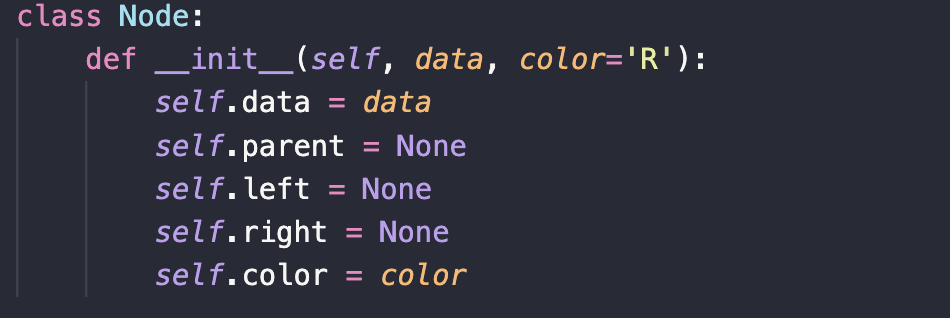
**(iv)** Right Left Case (s is right child of its parent and r is left child of s)

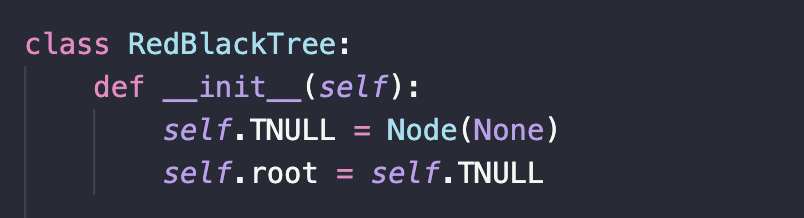
**(b)**: If sibling is black and its both children are black, perform recoloring, and recur for the parent if parent is black.

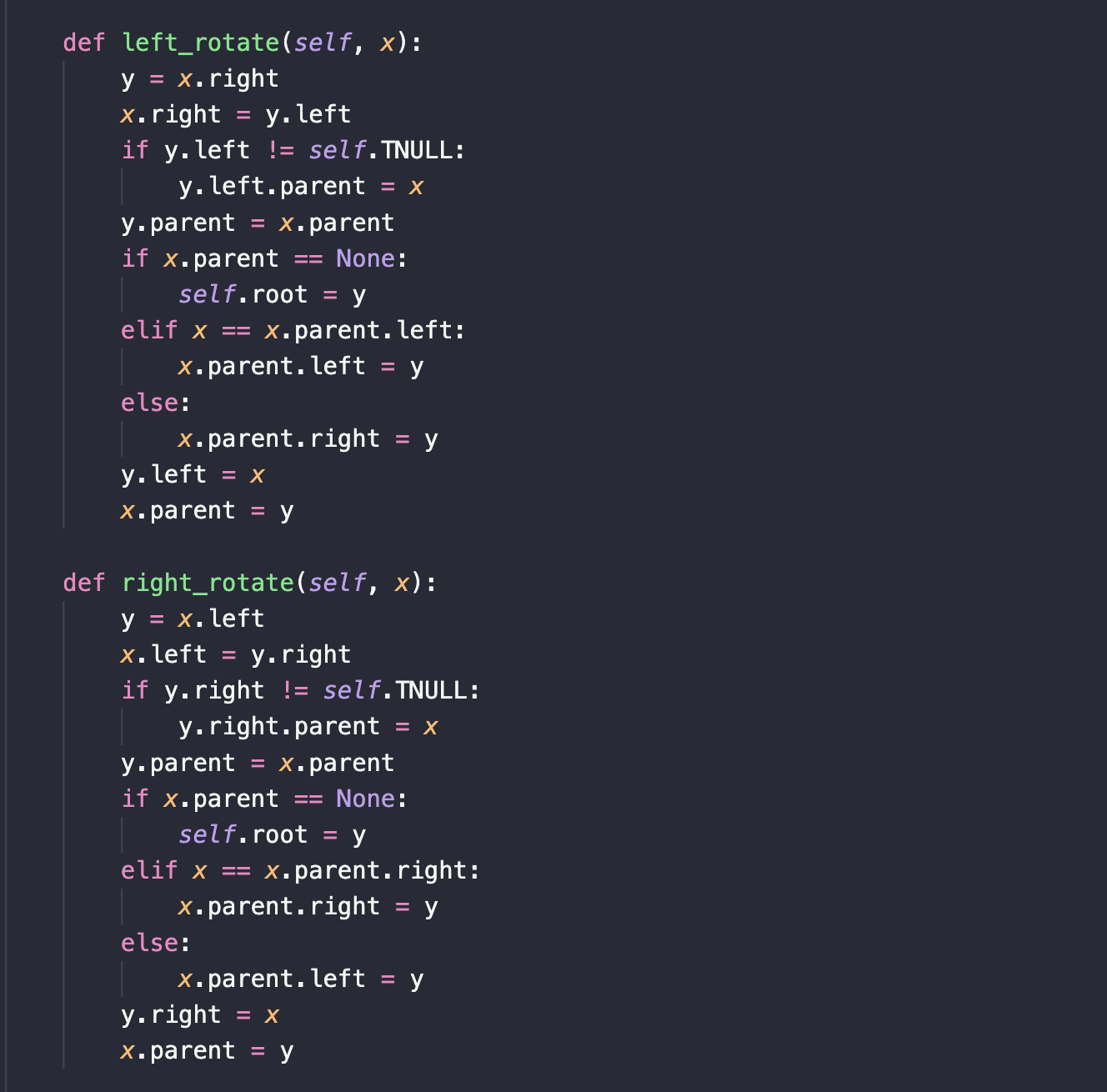
**(c)**: If sibling is red, perform a rotation to move old sibling up, recolor the old sibling and parent. The new sibling is always black (See the below diagram). This mainly converts the tree to black sibling case (by rotation) and leads to case (a) or (b). This case can be divided in two subcases.   
  
**(i)** Left Case (s is left child of its parent). This is mirror of right right case shown in below diagram. We right rotate the parent p.   
  
**(ii)** Right Case (s is right child of its parent). We left rotate the parent p.

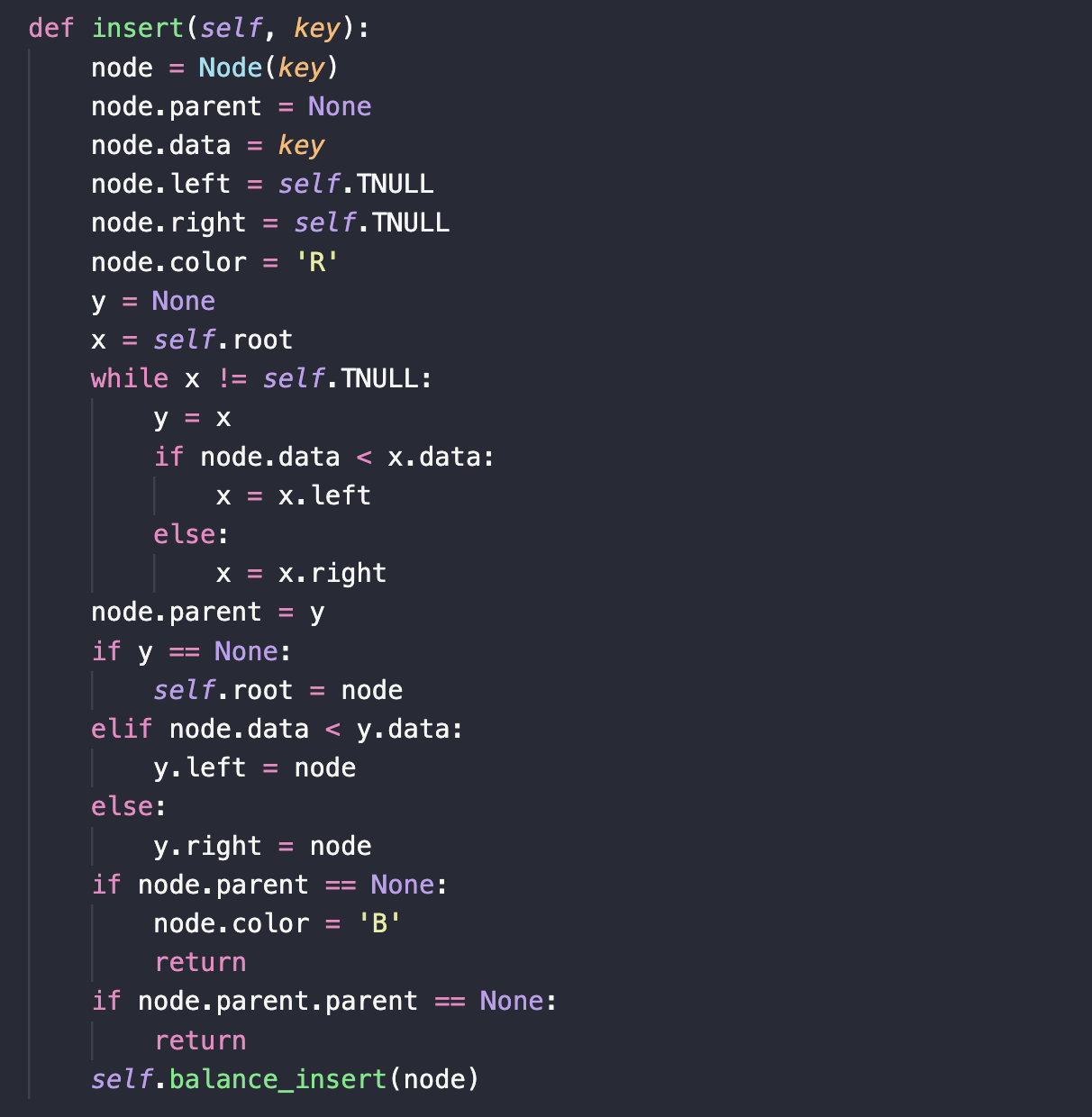
**3.3)** If u is root, make it single black and return (Black height of complete tree reduces by 1).

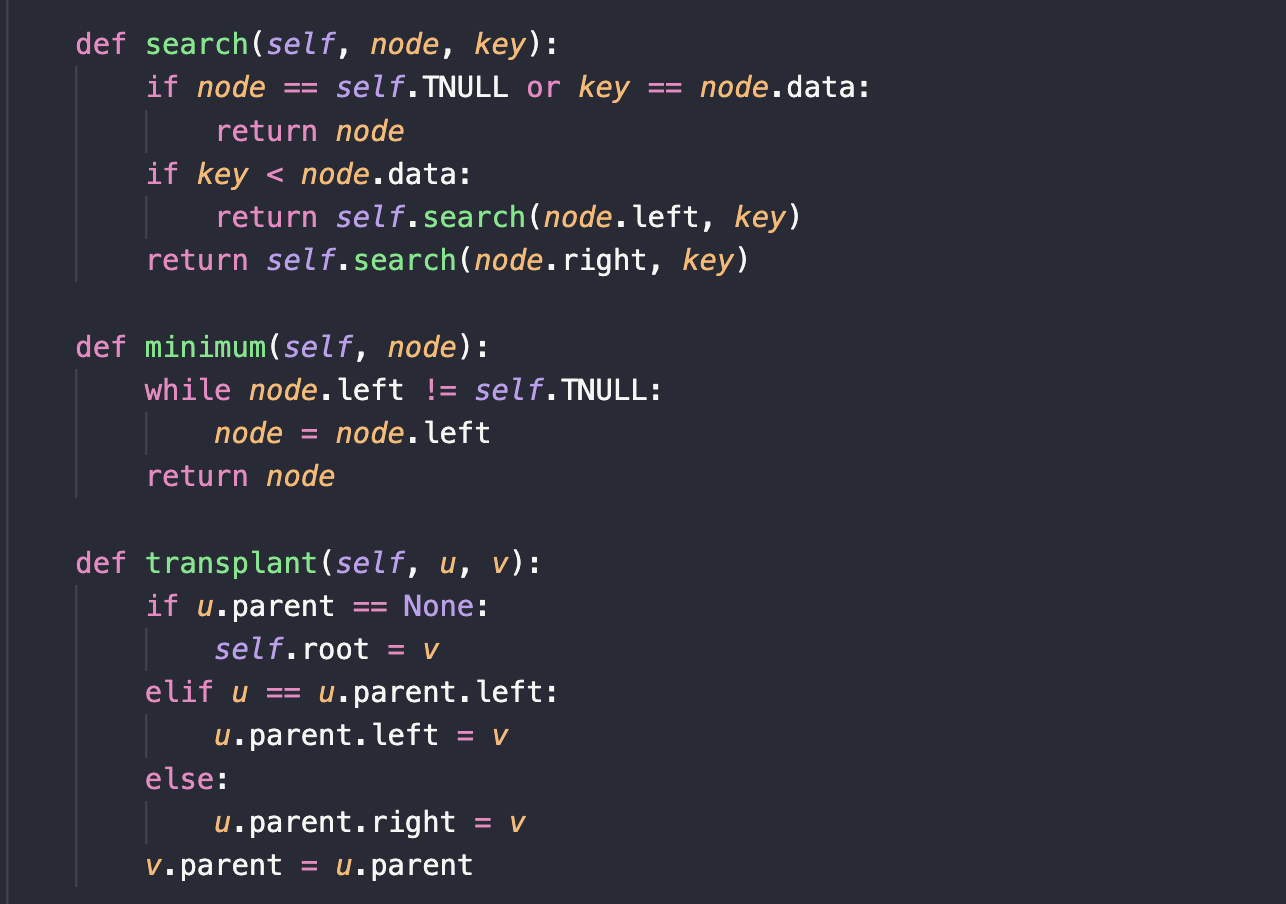








  
****

****

****