Red Black tree insertion

```
insert (const int & data)
{   Node *pt = new Node(data)
    root = BSTInsert (root, pt); // does a normale
                                    insertion in a BST

    // Now we need to check if red black tree
       properties are violated and fix them
    fixviolation (root, pt)
}
fixviolation (Node *&root, Node *&pt)
{   Node *parent_pt = NULL;
    Node *grand_parent_pt = NULL;
    while ((pt != root) && (pt -> color != black) &&
                pt -> parent -> color == RED))
    {   parent_pt = pt -> parent;
        grand_parent_pt = pt -> parent -> parent;
    // Case 1: parent of pt is left child of grand
       parent of pt.
    if (parent_pt == grandparent_pt -> left)
    {   Node * uncle_pt = grand_parent_pt -> right
        // Case 1a: The uncle of pt is also red
           only recoloring required
        if (uncle_pt != NULL && uncle_pt -> color
                                    == RED)
        {   grand_parent_pt -> color = RED;
            parent_pt -> color = BLACK;
            uncle_pt -> color = BLACK;
        }   pt = grand_parent_pt;
```

```
else
{
// Case 1b: pt is right child of its parent Left
            rotation required.
    if (pt == parent_pt->right)
    {
        rotateLeft(root, parent_pt);
        pt = parent_pt;
        parent_pt = pt->parent;
    }
// Case 1c: pt is Left child of its parent
        right rotation required.
    rotateRight(root, grand_parent_pt);
    swap(parent_pt->color, grand_parent_pt->color);
    pt = parent_pt;
}
// Case 2: Parent of pt is right child of grand
        parent of pt.
else
{
    Node *uncle_pt = grand_parent_pt->left;
// Case 2a: The uncle of pt is also red
            only recoloring required.
    if ((uncle_pt != NULL) && (uncle_pt->color == RED))
    {
        grand_parent_pt->color = RED;
        parent_pt->color = BLACK;
        uncle_pt->color = BLACK;
        pt = grand_parent_pt;
    }
    else
    { // Case 2b: pt is left child of its parent
            right rotation required
```

```
        if (pt == parent_pt -> left)
        {
            rotateRight(root, parent_pt);
            pt = parent_pt;
            parent_pt = pt -> parent;
        }
        // Case 2c : pt is right child of its parent, left
        rotation required
        rotateLeft(root, grand_parent_pt);
        swap(parent_pt -> color, grand_parent_pt -> color);
        pt = parent_pt;
    }
  }
 }
 root -> color = BLACK;
}
```