Prog-7 : Red Black trees insertion

Shikha.N
1BM18CS149
18-11-20

```
void RBTree :: insert (int data){
   Node * p = new Node (data);
   root = BSTInsert (root, p) ;//normal
   fixViolation( root, P);              BST insert.
   }

 // To fix the violations caused by BST insertion
void RBTree :: fixviolation (Node *root , Node * p)
{  Node * parent_pt=NULL;
     Node * grand_parent_pt= NULL ;
  while ((P != root) && (P -> color != B) &&
              (P -> parent -> color == R))
  {   parent_pt = p -> parent;
       grand-parent_pt = p -> parent -> parent ;
   // case A: parent of p is left child of G.P of p
  if (parent_pt == grand-parent_pt -> left)
  {  Node * uncle = grand-parent_pt -> right;
      if (uncle != NULL && uncle -> color == R)
      { grandparent_pt -> color = R ;
        parent_pt -> color = B ;
        uncle -> color = B .
        p = grand-parent-pt;
      }
```

Shikha

```
else
{ if (p= = parent_pt → right) {
   rotate left (root, parent_pt);
     p = parent -pt;
     parent -pt = p → parent;
   }

   rotate Right ( root , grand-parent -pt );
   Swap ( parent -pt → color, grand-parent -pt
                                   → color);

   p = parent - pt;

   }
 }
// case B : parent of p is right child of G.P of p
else {
   Node * uncle = grand-parent _pt → left;
   if ( uncle != NULL && uncle → color == R)
   { grandparent _pt → color = R;
     parent - pt → color = B;
     uncle → color = B
      p = grand -parent _ pt;

   }
  else
  { if ( p = = parent _ pt → left)
    { RotateRight (root, parent_pt);
       p = parent _pt;
       parent - pt = p → parent;
    } rotateLeft (root, grand-parent - pt);
Swap (parent -pt → color, grand -parent .pt → color);
   p = parent - pt; } } } root → color = B;
                      }
```

Shikha