VYSHNAVI.M.N
1BM18CS125

Red-Black Tree

```
enum COLOR { Red, Black };

typedef struct tree_node {
    int data;
    struct tree_node *right;
    struct tree_node *left;
    struct tree_node *parent;
    enum COLOR color;
} tree_node;

typedef struct red_black_tree {
    tree_node *root;
    tree_node *NIL;
} red_black_tree

tree_node* new_tree_node (int data) {
    tree_node* n = malloc (sizeof (tree_node));
    n->left = NULL;
    n->right = NULL;
    n->parent = NULL;
    n->data = data;
    n->color = Red;
   return n;
}
void left_rotate(red_black_tree *t, tree_node *x
{
   tree_node *y = x->right;
   x->right = y->left;
   if (y->left != t->NIL) {
        y->left->parent = x;
   }
```

```
    y -> parent = x -> parent;
    if (x -> parent == t -> NIL) {
        t -> root = y;
    }
    else if (x == x -> parent -> left) {
        x -> parent -> left = y;
    }
    else {
        x -> parent -> right = y;
    }
    y -> left = x;
    x -> parent = y;
}

void right_rotate (red_black_tree *t, tree_node
                                        *x) {

   tree_node *y = x -> left;
   x -> left = y -> right;
   if (y -> right != t -> NIL) {
       y -> right -> parent = x;
   }
   y -> parent = x -> parent;
   if (x -> parent == t -> NIL) {
       t -> root = y;
   }
   else if (x == x -> parent -> right) {
     x -> parent -> right = y;
   }
   else {
    x -> parent -> left = y;
   }
```

```
    y -> right = x;
    x -> parent = y;
}

void  insertion_fixup (red-black_tree *t, tree_node
                                              * z)
{
   while (z -> parent -> color == Red) {
      if (z -> parent == z -> parent -> parent -> left)
      {
         tree_node *y = z -> parent -> parent -> right;
        if (y -> color == Red) {
           z -> parent -> color = Black;
           y -> color = Black;
           z -> parent -> parent -> color = Red;
           z = z -> parent -> parent;
        }
        else {
           if (z == z -> parent -> right) {
             z = z -> parent;
             left_rotate (t, z);
           }
           z -> parent -> color = Black;
           z -> parent -> parent -> color = Red;
           right_rotate (t, z -> parent -> parent);
        }
     }
   else {
      tree_node *y = z -> parent -> parent -> left;
       if (y -> color == Red)
```

```
            t -> root        IL) {

          {
             z -> parent -> color = Black;
             y -> color = Black;
              z -> parent -> parent -> color = Red;
              z = z -> parent -> parent;
          }

      else {
        if (z == z -> parent -> left) {
           z = z -> parent;
           right_rotate (t, z);
        }
        z -> parent -> color = Black;
        z -> parent -> parent -> color = Red;
        left_rotate (t, z -> parent -> parent);
      }
     }
   }
    t -> root -> color = Black;
 }

void  insert (red_black_tree *t, tree_node *z)
{
   tree_node * y = t -> NIL;
   tree_node * temp = t -> root;
    while (temp != t -> NIL) {
       y = temp;
       if (z -> data < temp -> data)
           temp = temp -> left;
       else
           temp = temp -> right;
    }
    z -> parent = y;
    if (y == t->NIL) {
        t->root = z;
    }
    else if (z->data < y->data)
        y->left = z;
    else
        y->right = z;
    z->right = t->NIL;
    z->left = t->NIL;
    insertion_fixup(t, z);
}
```