DATA STRUCTURES & ALGORITHM MINIPROJECT

TOPIC: RED BLACK TRESS

Name- Vidisha Sharma & Sneha Gupta

Roll no-A046 / A016

Code-

```
#include<iostream>
using namespace std;
struct node_of_the_red_black_tree
{
  int key;
   node_of_the_red_black_tree *parent;
      char color;
      node_of_the_red_black_tree *left;
      node_of_the_red_black_tree *right;
};
class Red_Black_Tree
{
     node_of_the_red_black_tree *root;
     node_of_the_red_black_tree *temp_node1;
   public :
     Red_Black_Tree()
     {
           temp_node1=NULL;
           root=NULL;
     }
     void insert_a_node_to_rbt();
     void insert_a_node_to_rbtfix(node_of_the_red_black_tree *);
     void leftrotate_rbt_node(node_of_the_red_black_tree *);
```

```
    void rightrotate_rbt_node(node_of_the_red_black_tree *);
    void del_rbt_node();
    node_of_the_red_black_tree* successor(node_of_the_red_black_tree *);
    void del_rbt_nodefix(node_of_the_red_black_tree *);
    void disp();
    void display( node_of_the_red_black_tree *);
    void search_rbt_node();
};
void Red_Black_Tree::insert_a_node_to_rbt()
{
   int init_value,i=0;
   cout<<"\nKey value for the node_of_the_red_black_tree to insert_a_node_to_rbt in RBT: ";
cin>>init_value;
   node_of_the_red_black_tree *temp_node,*temp_node1;
   node_of_the_red_black_tree *t=new node_of_the_red_black_tree;
   t->key=init_value;
   t->left=NULL;
   t->right=NULL;
   t->color='r';
   temp_node=root;
   temp_node1=NULL;
   if(root==NULL)
   {
       root=t;
       t->parent=NULL;
   }
   else
   {
      while(temp_node!=NULL)
      {
          temp_node1=temp_node;
          if(temp_node->key<t->key)
             temp_node=temp_node->right;
```

```
            else
                temp_node=temp_node->left;
        }
        t->parent=temp_node1;
        if(temp_node1->key<t->key)
            temp_node1->right=t;
        else
            temp_node1->left=t;
    }
    insert_a_node_to_rbtfix(t);
}
void Red_Black_Tree::insert_a_node_to_rbtfix(node_of_the_red_black_tree *t)
{
    node_of_the_red_black_tree *u;
    if(root==t)
    {
        t->color='b';
        return;
    }
    while(t->parent!=NULL&&t->parent->color=='r')
    {
        node_of_the_red_black_tree *g=t->parent->parent;
        if(g->left==t->parent)
        {
                if(g->right!=NULL)
                {
                    u=g->right;
                    if(u->color=='r')
                    {
                        t->parent->color='b';
                        u->color='b';
                        g->color='r';
```

```
                t=g;
            }
        }
        else
        {
            if(t->parent->right==t)
            {
                t=t->parent;
                leftrotate_rbt_node(t);
            }
            t->parent->color='b';
            g->color='r';
            rightrotate_rbt_node(g);
        }
    }
    else
    {
        if(g->left!=NULL)
        {
            u=g->left;
            if(u->color=='r')
            {
                t->parent->color='b';
                u->color='b';
                g->color='r';
                t=g;
            }
        }
        else
        {
            if(t->parent->left==t)
            {
```

```
                    t=t->parent;
                    rightrotate_rbt_node(t);
int x;
cout<<"\nEnter the key of the node_of_the_red_black_tree to be del_rbt_nodeeted: "; cin>>x;
node_of_the_red_black_tree *temp_node;
temp_node=root;
node_of_the_red_black_tree *y=NULL;
node_of_the_red_black_tree *temp_node1=NULL;
int found=0;
while(temp_node!=NULL&&found==0)
{
    if(temp_node->key==x)
       found=1;
    if(found==0)
    {
        if(temp_node->key<x) temp_node=temp_node->right;
        else
          temp_node=temp_node->left;
    }
}
if(found==0)
{
     cout<<"\nElement Not Found.";
     return ;
}
else
{
   cout<<"\ndel_rbt_nodeeted Element: "<<temp_node->key;
   cout<<"\nColour: "; if(temp_node->color=='b')
cout<<"Black\n";
else
cout<<"Red\n"; if(temp_node->parent!=NULL)
```

```
        cout<<"\nParent: "<<temp_node->parent->key;
    else
        cout<<"\nno parent node_of_the_red_black_tree present "; if(temp_node->right!=NULL)
        cout<<"\nRight Child: "<<temp_node->right->key;
    else
        cout<<"\nno right child node_of_the_red_black_tree present. "; if(temp_node->left!=NULL)
        cout<<"\nLeft Child: "<<temp_node->left->key;
    else
        cout<<"\nno left child node_of_the_red_black_tree present.  ";
    cout<<"\nnode_of_the_red_black_tree del_rbt_nodeeted."; if(temp_node->left==NULL||temp_node->right==NULL)
       y=temp_node;
    else
       y=successor(temp_node);
    if(y->left!=NULL)
       temp_node1=y->left;
    else
    {
       if(y->right!=NULL)
          temp_node1=y->right;
       else
          temp_node1=NULL;
    }
    if(temp_node1!=NULL)
       temp_node1->parent=y->parent;
    if(y->parent==NULL)
       root=temp_node1;
    else
    {
      if(y==y->parent->left)
        y->parent->left=temp_node1;
      else
        y->parent->right=temp_node1;
```

```
        }
        if(y!=temp_node)
        {
           temp_node->color=y->color;
           temp_node->key=y->key;
        }
        if(y->color=='b')
           del_rbt_nodefix(temp_node1);
     }
}

void Red_Black_Tree::del_rbt_nodefix(node_of_the_red_black_tree *temp_node)
{
   node_of_the_red_black_tree *s;
   while(temp_node!=root&&temp_node->color=='b')
   {
        if(temp_node->parent->left==temp_node)
        {
             s=temp_node->parent->right;
             if(s->color=='r')
             {
                  s->color='b';
                  temp_node->parent->color='r';
                  leftrotate_rbt_node(temp_node->parent);
                  s=temp_node->parent->right;
             }
             if(s->right->color=='b'&&s->left->color=='b')
             {
                  s->color='r';
                  temp_node=temp_node->parent;
             }
             else
```

```
        {
            if(s->right->color=='b')
            {
                s->left->color=='b';
                s->color='r';
                rightrotate_rbt_node(s);
                s=temp_node->parent->right;
            }
            s->color=temp_node->parent->color;
            temp_node->parent->color='b';
            s->right->color='b';
            leftrotate_rbt_node(temp_node->parent);
            temp_node=root;
        }
    }
    else
    {
        s=temp_node->parent->left;
        if(s->color=='r')
        {
            s->color='b';
            temp_node->parent->color='r';
            rightrotate_rbt_node(temp_node->parent);
            s=temp_node->parent->left;
        }
        if(s->left->color=='b'&&s->right->color=='b')
        {
            s->color='r';
            temp_node=temp_node->parent;
        }
        else
        {
```

```
                if(s->left->color=='b')
                {
                    s->right->color='b';
                    s->color='r';
                    leftrotate_rbt_node(s);
                    s=temp_node->parent->left;
                }
                s->color=temp_node->parent->color;
                temp_node->parent->color='b';
                s->left->color='b';
                rightrotate_rbt_node(temp_node->parent);
                temp_node=root;
            }
        }
     temp_node->color='b';
     root->color='b';
   }
}

void Red_Black_Tree::leftrotate_rbt_node(node_of_the_red_black_tree *temp_node)
{
    if(temp_node->right==NULL)
        return ;
    else
    {
        node_of_the_red_black_tree *y=temp_node->right;
        if(y->left!=NULL)
        {
            temp_node->right=y->left;
            y->left->parent=temp_node;
        }
        else
```

```
            temp_node->right=NULL;
        if(temp_node->parent!=NULL)
            y->parent=temp_node->parent;
        if(temp_node->parent==NULL)
            root=y;
        else
        {
           if(temp_node==temp_node->parent->left)
                temp_node->parent->left=y;
           else
                temp_node->parent->right=y;
        }
        y->left=temp_node;
        temp_node->parent=y;
    }
}
void Red_Black_Tree::rightrotate_rbt_node(node_of_the_red_black_tree *temp_node)
{
    if(temp_node->left==NULL)
        return ;
    else
    {
       node_of_the_red_black_tree *y=temp_node->left;
       if(y->right!=NULL)
       {
            temp_node->left=y->right;
            y->right->parent=temp_node;
       }
       else
            temp_node->left=NULL;
       if(temp_node->parent!=NULL)
            y->parent=temp_node->parent;
```

```
        if(temp_node->parent==NULL)
             root=y;
        else
        {
           if(temp_node==temp_node->parent->left)
                temp_node->parent->left=y;
           else
                temp_node->parent->right=y;
        }
        y->right=temp_node;
        temp_node->parent=y;
    }
}

node_of_the_red_black_tree*                Red_Black_Tree::successor(node_of_the_red_black_tree
*temp_node)
{
     node_of_the_red_black_tree *y=NULL;
    if(temp_node->left!=NULL)
    {
       y=temp_node->left;
       while(y->right!=NULL)
            y=y->right;
    }
    else
    {
       y=temp_node->right;
       while(y->left!=NULL)
            y=y->left;
    }
    return y;
}
```

```
void Red_Black_Tree::disp()
{
    display(root);
}
void Red_Black_Tree::display(node_of_the_red_black_tree *temp_node)
{
    if(root==NULL)
    {
        cout<<"\nEmpty Tree.";
        return ;
    }
    if(temp_node!=NULL)
    {
            cout<<"\n\t node_of_the_red_black_tree: ";
            cout<<"\n Key: "<<temp_node->key;
            cout<<"\n Colour: "; if(temp_node->color=='b')
    cout<<"Black";
   else
    cout<<"Red"; if(temp_node->parent!=NULL)
                cout<<"\n Parent: "<<temp_node->parent->key;
            else
                cout<<"\n no parent node_of_the_red_black_tree present "; if(temp_node->right!=NULL)
                cout<<"\n Right Child: "<<temp_node->right->key;
            else
                cout<<"\n no right child node_of_the_red_black_tree present. "; if(temp_node->left!=NULL)
                cout<<"\n Left Child: "<<temp_node->left->key;
            else
                cout<<"\n no left child node_of_the_red_black_tree present.  ";
            cout<<endl; if(temp_node->left)
   {
             cout<<"\n\nLeft:\n"; display(temp_node->left);
```

```
    }
    /*else
     cout<<"\nNo Left Child.\n";*/ if(temp_node->right)
    {
     cout<<"\n\nRight:\n"; display(temp_node->right);
    }
    /*else
     cout<<"\nNo Right Child.\n"*/
     }
}
void Red_Black_Tree::search_rbt_node()
{
    if(root==NULL)
    {
        cout<<"\nEmpty Tree\n" ;
        return  ;
    }
    int x;
    cout<<"\n Enter key of the node_of_the_red_black_tree to be search_rbt_nodeed: "; cin>>x;
    node_of_the_red_black_tree *temp_node=root;
    int found=0;
    while(temp_node!=NULL&& found==0)
    {
         if(temp_node->key==x)
            found=1;
         if(found==0)
         {
             if(temp_node->key<x) temp_node=temp_node->right;
             else
                 temp_node=temp_node->left;
         }
    }
```

```
    if(found==0)
        cout<<"\nElement Not Found.";
    else
    {
            cout<<"\n\t FOUND node_of_the_red_black_tree: ";
            cout<<"\n Key: "<<temp_node->key;
            cout<<"\n Colour: "; if(temp_node->color=='b')
    cout<<"Black";
   else
    cout<<"Red"; if(temp_node->parent!=NULL)
                cout<<"\n Parent: "<<temp_node->parent->key;
            else
                cout<<"\n no parent node_of_the_red_black_tree present"; if(temp_node->right!=NULL)
                cout<<"\n Right Child: "<<temp_node->right->key;
            else
                cout<<"\n no right child node_of_the_red_black_tree present. "; if(temp_node->left!=NULL)
                cout<<"\n Left Child: "<<temp_node->left->key;
            else
                cout<<"\n no left child node_of_the_red_black_tree present";
            cout<<endl;

    }
}
int main()
{
   int ch,y=0;
   Red_Black_Tree obj;
   do
   {
            cout<<"\nThe options list for Red Black Tree::" ;
            cout<<"\n1. To add a new node_of_the_red_black_tree in the Red-Black Tree";
```

```
        cout<<"\n2. To search_rbt_node the Red-Black Tree for a node_of_the_red_black_tree";
        cout<<"\n3. To del_rbt_nodeete an existing Red-Black tree node_of_the_red_black_tree";
        cout<<"\n4. To display all the node_of_the_red_black_trees of the Red-Black Tree ";
        cout<<"\n5. To exit the code execution." ;
        cout<<"\nInput: ";
        cin>>ch;
        switch(ch)
        {
            case 1 : obj.insert_a_node_to_rbt();
                     cout<<"\nNew node_of_the_red_black_tree added.\n";
                     break;
            case 2 :
                    obj.search_rbt_node();
                     break;
            case 3 :

                      obj.del_rbt_node();
                     break;
            case 4 : obj.disp();
                     break;
            case 5 : y=1;
                     break;
            default : cout<<"\nEnter a Valid Choice.";
        }
        cout<<endl;

   }while(y!=1);
   return 0;
}
```

Output-

To insert values in the trees-

```
The options list for Red Black Tree::
1. To add a new node_of_the_red_black_tree in the Red-Black Tree
2. To search_rbt_node the Red-Black Tree for a node_of_the_red_black_tree
3. To del_rbt_nodeete an existing Red-Black tree node_of_the_red_black_tree
4. To display all the node_of_the_red_black_trees of the Red-Black Tree
5. To exit the code execution.
Input: 1

Key value for the node_of_the_red_black_tree to insert_a_node_to_rbt in RBT: 12

New node_of_the_red_black_tree added.


The options list for Red Black Tree::
1. To add a new node_of_the_red_black_tree in the Red-Black Tree
2. To search_rbt_node the Red-Black Tree for a node_of_the_red_black_tree
3. To del_rbt_nodeete an existing Red-Black tree node_of_the_red_black_tree
4. To display all the node_of_the_red_black_trees of the Red-Black Tree
5. To exit the code execution.
Input: 1

Key value for the node_of_the_red_black_tree to insert_a_node_to_rbt in RBT: 45

New node_of_the_red_black_tree added.


The options list for Red Black Tree::
1. To add a new node_of_the_red_black_tree in the Red-Black Tree
2. To search_rbt_node the Red-Black Tree for a node_of_the_red_black_tree
3. To del_rbt_nodeete an existing Red-Black tree node_of_the_red_black_tree
4. To display all the node_of_the_red_black_trees of the Red-Black Tree
5. To exit the code execution.
Input: 1

Key value for the node_of_the_red_black_tree to insert_a_node_to_rbt in RBT: 23

New node_of_the_red_black_tree added.


The options list for Red Black Tree::
1. To add a new node_of_the_red_black_tree in the Red-Black Tree
2. To search_rbt_node the Red-Black Tree for a node_of_the_red_black_tree
3. To del_rbt_nodeete an existing Red-Black tree node_of_the_red_black_tree
4. To display all the node_of_the_red_black_trees of the Red-Black Tree
5. To exit the code execution.
Input: 1

Key value for the node_of_the_red_black_tree to insert_a_node_to_rbt in RBT: 98
```

To display values of trees-

```
The options list for Red Black Tree::
1. To add a new node_of_the_red_black_tree in the Red-Black Tree
2. To search_rbt_node the Red-Black Tree for a node_of_the_red_black_tree
3. To del_rbt_nodeete an existing Red-Black tree node_of_the_red_black_tree
4. To display all the node_of_the_red_black_trees of the Red-Black Tree
5. To exit the code execution.
Input: 4

        node_of_the_red_black_tree:
 Key: 23
 Colour: Black
 Parent: 12
 Right Child: 45
 Left Child: 12


Left:

        node_of_the_red_black_tree:
 Key: 12
 Colour: Black
 Parent: 23
 no right child node_of_the_red_black_tree present.
 no left child node_of_the_red_black_tree present.


Right:

        node_of_the_red_black_tree:
 Key: 45
 Colour: Black
 Parent: 23
 Right Child: 98
 no left child node_of_the_red_black_tree present.


Right:

        node_of_the_red_black_tree:
 Key: 98
 Colour: Red
 Parent: 45
 no right child node_of_the_red_black_tree present.
 no left child node_of_the_red_black_tree present.
```

To delete an existing node of a tree-

```
The options list for Red Black Tree::
1. To add a new node_of_the_red_black_tree in the Red-Black Tree
2. To search_rbt_node the Red-Black Tree for a node_of_the_red_black_tree
3. To del_rbt_nodeete an existing Red-Black tree node_of_the_red_black_tree
4. To display all the node_of_the_red_black_trees of the Red-Black Tree
5. To exit the code execution.
Input:
3

Enter the key of the node_of_the_red_black_tree to be del_rbt_nodeeted: 45

del_rbt_nodeeted Element: 45
Colour: Black

Parent: 23
Right Child: 98
no left child node_of_the_red_black_tree present.
node_of_the_red_black_tree del_rbt_nodeeted.
```

To exit the program-

```
The options list for Red Black Tree::
1. To add a new node_of_the_red_black_tree in the Red-Black Tree
2. To search_rbt_node the Red-Black Tree for a node_of_the_red_black_tree
3. To del_rbt_nodeete an existing Red-Black tree node_of_the_red_black_tree
4. To display all the node_of_the_red_black_trees of the Red-Black Tree
5. To exit the code execution.
Input: 5


--------------------------------
Process exited after 225.4 seconds with return value 0
Press any key to continue . . .
```